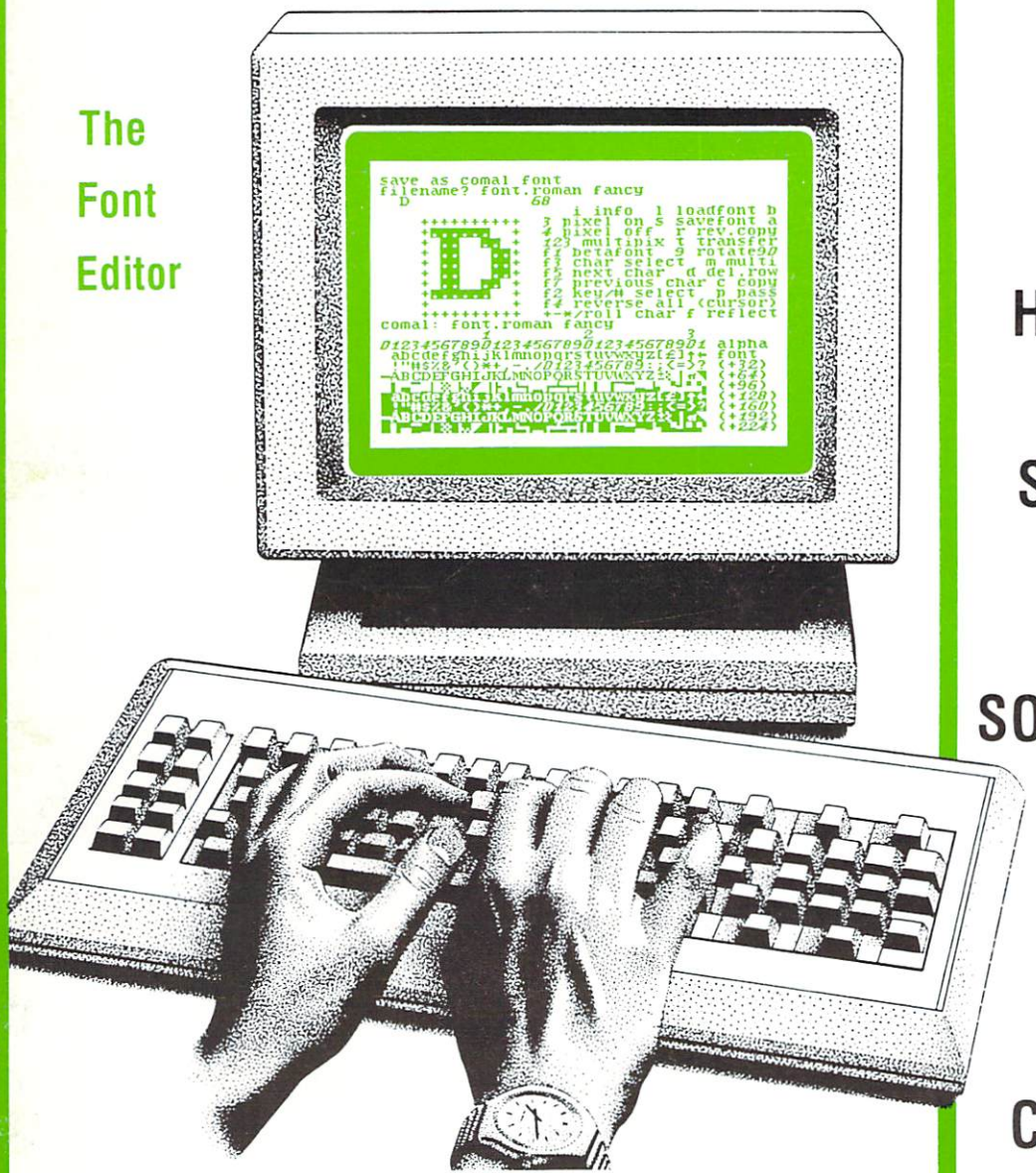
COMAL TODAY 10

comal is a programmers paradise

See Program on Page 78

The Font Editor



COMPACT BITMAPS
page 68

HOW TO DRAW
page 50

SCREENDUMP PACKAGE
page 72

SORTS IN COMAL
page 18

EXPERT SYSTEMS
page 16

COMAL CLINIC
page 29

FONTS
page 30

ALSO —

New Packages

Ready-To-Type Programs

Sorting Techniques

BULK RATE
U.S.POSTAGE
PAID
MADISON, WI
PERMIT 2981

IF YOUR LABEL SAYS
LAST ISSUE: 10
YOU MUST RENEW NOW.
USE ORDER FORM INSIDE

COMAL TODAY ISSUE 10 November 26, 1985

PUBLISHER

COMAL Users Group,
USA, Limited
6041 Monona Drive
Madison, WI 53716
608-222-4432

From the Editor's Disk

COMAL is no longer just for programmers. We are building quite a library of ready to LOAD and GO programs, many for the COMAL 2.0 Cartridge. TODAY Disk #9 contained a Package Maker program and a Print Shop Icon Editor. TODAY Disk #10 includes an expanded and LINKable Monitor Package, a Font Editor and Picture Compactor. No wonder Cartridge owners are becoming disk subscribers. Plus as a bonus, each TODAY Disk will now include ready to LINK and USE packages. We plan to include a popup menu system for Cartridge users. A similar system for the IBM PC sold thousands of copies at $50 each - ours will be a part of TODAY Disk #11.

COMAL 0.14 users are NOT being left out! The FONT Disk is now for use with BOTH COMAL 2.0 and 0.14. So is the Compacted Picture system on TODAY Disk #10. And we hope to include on TODAY Disk #11 a complete graphics editing system - just for COMAL 0.14.

This may be the last chance to subscribe to the TODAY Disks at the special $29.95 price. You can start your subscription with any disk. For example, a Cartridge user may wish to start with Disk #6 (the first double sided TODAY Disk). A 10 disk subscription would then provide disks 6-15 and cost $29.95 for the first 6 and $20 for the next 4 - total of only $49.95 (for 20 disk sides!).

Teachers will be happy to know that APPLE COMAL will soon be a reality. We are in the final stages of negotiations with UniCOMAL now. That means educational materials will be needed. We will help distribute these materials as they become available. Amiga COMAL is also a goal.

The reality of dealing with products imported from overseas has struck. The bad news is that the value of the dollar in Denmark has taken a steep fall. The dollar value dropped about 30%. Yes, that means we must pay 30% more for the items we import. Need I say more? Price changes must take effect immediately (IBM PC COMAL and C64 COMAL 2.0 Cartridge). If the dollar continues its downward trend, the prices may even go higher. For once, those who bought early got a good deal! If the price goes down later, subscribers have our Price Protection Plan.

By the end of January we should be shipping several brand new items. TYPING Disk will include Typing Tutor, a spelling dictionary of thousands of words, and several Dvorak Keyboard programs. Complete documentation will be included on the disk along with a program that will display it to the screen or printer. Our first COMAL GAMES Disk will be for both COMAL 0.14 and 2.0. You can order either of them now for delivery when released - only $9.75 each to subscribers. Packages Library should also be ready then - $19.95 for a double disk with book.

Last issue we promised to REVEAL how to list unlistable lines in ROD THE ROADMAN. Just type in this command: REVEAL. Now, how would you HIDEAWAY the lines again? Watch next issue for the answer.

We are trying a different way of listing programs. Tell us how you like it. See page 43 for an example.

We hope you continue giving a copy of COMAL disks to your friends and schools. This includes all our TODAY Disks and User Group Disks, as well as the Tutorial Disk and Auto Run Demo Disk. But please do not give out copies of our special collection disks or disks that are part of book sets.

COMALites Unite!

by David Stidolph

February 8-9, 1986 will be THE COMMODORE SHOW II in San Francisco. This will be the first major computer show I have ever attended, so I'd like to meet as many **COMALites** as possible. I will be helping Captain COMAL with the COMAL seminar at the show.

"Who am I?" you ask? Well, I am a COMAL programmer and I help edit the programs that are sent to COMAL Users Group. I got involved with COMAL in November 1983 with the release of COMAL 0.14 for the Commodore 64. It was such an improvement over BASIC and every other language I had used that I dropped everything and began writing exclusively in COMAL (well - maybe a little assembly code to make things go faster). I have watched the language grow and adapt itself to many tasks.

Some people ask why some programs are not listed in the newletter. Every program mentioned in the newsletter is on its matching TODAY Disk. By putting the programs on the disk, and not listing them, COMAL TODAY can provide a maximum amount of information about how the programs work, and not simply on how they are coded. TODAY Disks are double sided, the front for COMAL 0.14 and the back for COMAL 2.0. You can subscribe to the disk just like you subscribe to the newsletter.

TODAY Disk #10 features some games like BOGGLE and COMAL ACE (a WW-I biplane game), a nuclear bomb simulation called ATOM'HAPPY, sprite shapes, pictures, fonts, useful procedures, and functions. All the programs discussed throughout this issue are also on the disk. PROC.REVEAL is included. This is the procedure built into the ROD THE ROADMAN program on TODAY DISK #9.

We hope you will subscribe to both COMAL TODAY and the TODAY Disks. This will save you from typing the programs in. Starting with this issue, the directory listing for the matching TODAY DISK will be in the following issue.

One of the features of the COMAL 2.0 cartridge is the addition of PACKAGES. COMAL Packages help to expand the language without creating multiple "versions" of the language (i.e. no COMAL Ver. 23.1a4).

The art of writing packages was a complete mystery until Jesse Knight started exploring the cartridge and examining how the built-in packages were formed. He worked for over a month until Len Lindsay came back from a trip to Denmark with 18 photo copied pages from the upcoming Commodore Tutorial Binder. Armed with this extra information Jesse wrote the book COMAL 2.0 PACKAGES. This book answered many of the questions about how to write packages, but left some unanswered.

Since Jesse's COMAL 2.0 PACKAGES book was released, many packages have been written, and more information on packages has been gathered. To further ease the developement of more packages, I am putting together the second 2.0 Packages book/disk combo, tentatively named **PACKAGES LIBRARY**. The book will be designed to explain all the packages found on the double sided disk. It also will aid anyone wishing to write their own packages. In it, I will seek to explain some of the least known features of Commodore's Editor/Assembler.

The book/disk set should be available in late January 1986. You can order a copy now for $19.95 (plus $2 shipping) and your copy will be shipped to you as soon as it is released. ■

How to Type in COMAL Programs

Line numbers are irrelevant to a running COMAL program. COMAL only provides line numbers for your benefit in editing the program. Thus most magazines do not use line numbers when listing a COMAL program. It is up to YOU to provide the line numbers. But of course, COMAL can do it for you quite easily. Just follow these steps to type in a COMAL program:

1) Enter command: NEW
2) Enter command: AUTO
3) Type in the program
4) When done:
   Version 0.14: Hit <RETURN> key twice
   Version 2.0 : Hit <STOP> key

Remember - use unshifted letters thoughout entering the program. If letters are capitalized in the listing it does not mean to use SHIFT with those letters. They are capitalized merely to be easy to read. The only place to use SHIFTED letters is inside quotes. Also, you don't have to type leading spaces in a line. They are listed only to emphasize structures. You DO have to type a space between COMAL words in the program.

LONG PROGRAM LINES: We are continuing to print COMAL TODAY with two columns per page, printed in elite (12 pitch) with 42 characters maximum per line. This makes it easiest to read. However, some program listings have program lines that extend beyond the 42 character limit. We decided to list these lines in the same manner that COMAL uses when listing long lines on a 40 column screen. We simply break the line right at the 42 character spot, and continue it on the next line, indenting it properly to keep the program structures obvious. These are called wrap lines. To draw your attention to these continued lines we add a //wrap line comment to the end of the line. Whenever you see this make sure you type both lines as one continuous program line! The following example includes a line with more than 42 characters that we must list on two lines, but you must type in as one long program line:

```
IF CURRENT'NAME$<>"FINISH" THEN PRINT'LABE
L(CURRENT'NAME$,PHONE$) //wrap line
```

If you type in this long program line as two shorter program lines, COMAL will not object (although sometimes it will)! But, the program will not work unless it is entered as one long line. The procedure name PRINT'LABEL is split onto two lines in the listing, but the //wrap line draws your attention to this fact. ■

Letters

Colin,

I'd like to say that being unknown is probably COMAL's biggest obstacle to acceptance in the USA. I've been trying to spread COMAL on the Central Coast of California, as well as in Central Oregon, and have been talking to teachers in those areas. Every teacher I've talked to has heard of BASIC; none have heard of COMAL. Some ask, if its such a great language, why don't they see articles about it in computer magazines?

According to Commodore, COMAL is now the third most popular programming language on the C64, but I'll be the first to admit that I have only seen one or two COMAL reviews in computer magazines. Since the Commodore computer base seems to be fairly large, COMAL would benefit from reviews, advertisements, programming tips, and programs published in computer magazines dedicated to Commodore computers. This would be a useful way to give COMAL the publicity it deserves. And what if COMAL TODAY 'went public' and was displayed on store magazine racks nationwide? Perhaps someday people will say they haven't heard of BASIC, just COMAL. And when you think about it, thats just as it should be.

Tom Dipp
California

Tom,

You bring up some obvious points regarding COMAL's popularity in the US (and CANADA). While we'd like to see COMAL grow in popularity, we realize the COMAL Users Group can't do it all. If we all pitch in and do our part to spread the word, COMAL will gain the respect and acceptance it so rightfully deserves. That means that those of us capable of writing articles for the magazine industry must do so. All of the major Commodore magazines with one exception will accept well written COMAL articles/programs.

If you want to read COMAL articles in the popular Commodore magazines, write to the Editors and let them know how you feel. When I was the Tech Editor at Commander Magazine, I paid close attention to the wants and needs of my readers, as expressed through their letters and phone calls.

Since COMAL is an educational language, we find it odd that it has not caught on better in our schools. There are at least two problems I see.

1) BASIC is free, and worth every penny of it.
2) COMAL for the Apple is not yet available.

Those teachers that have C-64s in the classroom, and have seen COMAL, jump on our bandwagon with a fervor approaching that of a Saturday night revival meeting.

We here at the COMAL Users Group are doing all we can to get an Apple COMAL. I suggest that the COMALites of this country do what they can to spread some COMAL goodwill by giving demonstrations at user group meetings and starting COMAL SIGs.

There are no plans for COMAL TODAY to be sold over the counter. Given time and dedication, someday BASIC and COMAL will trade places in the popularity polls. ■

Letters

Dear Mr. Lindsay,

The adolescent program of our Psychoeducational Program here in Augusta consists of classes in four schools covering three countries. We have a Commodore 64 computer system in each of these classrooms which we use in three ways. First we use the games programs as a reward for appropriate behavior. Second we use the spreadsheets and programs that I have written for our monthly reporting. Last, but not least, I teach programming to the students.

Until recently, I wrote our programs and instructed in BASIC. About two months ago I sent for the introductory COMAL package and have since replaced BASIC entirely with COMAL.

COMAL is easy to learn and is a self-documenting language with good structure. Our students have special needs and require structure in their daily lives. COMAL fits into our program by helping them to learn the benefits of structure in planning and implementing computer programs. (Of course they also like the turtle which serves as a reward).

In the issue of COMAL TODAY that I received with my package there was an offer for a free subscription to COMAL TODAY. I would like to apply for this subscription. Also, any discount on products that may be available to us would be appreciated.

Sincerely

H. L. Pond, JR, MSW
Sand Hills Psychoeducational Program
Augusta, Georgia

Dear Mr. Pond,

We're pleased that you have put COMAL to work in your classroom. Your letter is published here so that other progressive educators may learn of the special assistance we provide to teachers.

Any instructor teaching COMAL is entitled to a free subscription to COMAL TODAY. Please make the request on school letterhead. We also offer quantity discounts on our COMAL products so that teachers and their administrators may easily afford to use COMAL in the classroom. ■

Dear Calvin COMAL - This is a short note to you on how COMAL is doing here in West Germany. The September issue of 64'er, the most popular Commodore 64 magazine in Germany, featured a three page article about C64 COMAL which calls COMAL *"The universal programming language."* Here is a translation of part of the article:

Forget everything you know about programming languages for your 64. COMAL 80 will transform your C64 into a whole new computer! Programming in COMAL is so easy and comfortable that one could believe to be working with a whole new computer. Every drawback or bug of the 64 has been eliminated through COMAL and if somebody should find something worth improving, no problem. COMAL's package capabilities allow expansions for every task. Summary: COMAL 80 is not a language, but THE language for the C64.

We will see what the future will bring - Patrick Roye, American Red Cross

Thanks, Patrick. COMAL was designed right from the start to be the replacement for the outdated BASIC. I guess the Germans agree! ■

Questions and Answers

QUESTION: I am trying to complete exercise #4 in the Cartridge Tutorial Binder on page 81. I simplified the program and got the error "wrong type of RETURN". I can make the program work if I do not use FUNC and ENDFUNC. If I tell FUNC to RETURN any number it will work. Why is a number and not a string expected? - Janice Barton, Topeka, Kansas

ANSWER: The problem seems to be a misunderstanding of what a FUNCtion is. A function can do everything a PROCedure does BUT it returns a value. Also, you call a PROCedure, but you do not call a function. You use a function just like a variable (only the function calculates itself each time you use it).

On page 81 of the Cartridge Tutorial Binder they are using a string function! You tried using a numeric function. A string function RETURNs a string. A numeric function RETURNs a number. The name of your function defines what type it is. If it ends with a $ it is a string function. If it ends with a # it is a numeric function (integer result only). If it is just a simple name it is numeric. For example:

```
FUNC COUNT# // an integer numeric function
FUNC COST   // a real number function
FUNC PLAYER$// a string function
```

Try this modification of your function that will return a 5 character long string filled with the character you pass to it:

```
NEW
AUTO
FUNC fillup$(char$) CLOSED
  text$="" // nothing yet
  FOR x=1 TO 5 DO text$:+char$
  RETURN text$
ENDFUNC fillup$
```

Type that in (press the STOP key after the last line). Then type the command:

```
SCAN
```

Now, in direct mode type this:

```
PRINT fillup$("*")
```

COMAL responds like this:

```
*****
```

QUESTION: Are you allowed to nest procedure definitions?

ANSWER: In COMAL 2.0, yes. Nested procedures may be used to help organize procedures to be used by several programs. For example:

```
PROC start
  FOR x=1 TO 3 DO star(x)
  PROC star(num)
    FOR num'star=1 TO num DO PRINT "*",
    PRINT // carriage return
  ENDPROC star
ENDPROC start
```

Now, just LIST the procedure START to disk so it can be merged with other programs. Of course the STAR procedure definition would go along with it!

QUESTION:
How do I LIST my COMAL 0.14 program to the printer in upper/lower case mode? I know I must set the secondary address to 7, but I don't know how to do it.

ANSWER:

```
   OPEN FILE 255,"", UNIT 4,7, WRITE
   SELECT "LP:"
   LIST
```

Remember to SELECT "DS:" *after the printing stops.* ■

Bug Fixes

The ILLUSION program that was listed in COMAL TODAY #8 seems to be jinxed. One correction to it was published in COMAL TODAY #9, and now we have some more. If you try to convert it to COMAL 2.0, please make these corrections on page 32:

In the proc **getdata**, the calls to the proc **getarray** must be changed. COMAL 2.0 requires a set of empty parens to follow an array name in a parameter list.

```
getarray(xv#,11)    // old, incorrect
getarray(xv#(),11) // new, correct
```

Make this correction to all eight calls to getarray in the proc getdata. Also, remove the CLOSED from the proc **draw**.

In case you missed the correction in COMAL TODAY #9, here it is:

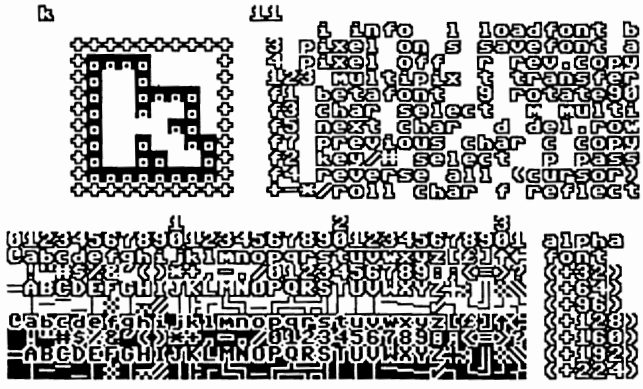
```
//
init'graph // add this line
block2 // main program
```

On TODAY Disk #9 the program **OKI92'TEST** had an error. To correct it, LOAD the program, then issue this command in the immediate mode:

```
CHANGE "'","<leftarrow>"
```

Where <leftarrow> is the left arrow key at the top left side of the keyboard. This key produces an underline character when pressed.

Hold the RETURN key down until all the changes are made, then DELETE and SAVE the program. ■



Look Back Before It's Too Late

by Captain COMAL

If you are just starting with COMAL (or if you want to know **everything** about COMAL) you should be very interested in all the back issues of COMAL TODAY. The problem is that we don't have enough back issues available for everybody. So get your copies now before they are gone. The first four issues were gone nearly a year before we finally got them reprinted (as a spiral bound book named COMAL YESTERDAY).

COMAL YESTERDAY includes everything from the first four issues of COMAL TODAY. I'm not kidding when I tell you that there are hundreds of tips and notes about using COMAL in there. It also contains plenty of complete COMAL 0.14 program listings ready for you to try out. Plus many articles aimed at making things a little easier to understand. Here is a list of just some of the things:

COMAL In School
Is 10K Enough?
Programming Languages For Beginners
Sprites And COMAL
Joystick, Paddle, Koala, Light Pen Control
INPUT From The Graphics Screen
Recursion
Teaching COMAL In A Classroom
Printer Notes
Parameters
It's The Little Things That Count
Learning COMAL
Background On Random Files
Just For Very Beginners
How To Dump A Graphic Screen To Printer
Spiralateral
COMAL 0.14 Name Table
Disk Talk

It only gets better with each issue after that. Remember, this is only a small partial listing of the major items!

COMAL TODAY ISSUE #5

Just For Beginners
COMAL 2.0 Input Field
How COMAL Statements Are Stored
COMAL 0.14 Keyword Table
Fun With COMAL 2.0
Attack Of The Mutant COMALs
COMAL 0.14 - Just For Fun
Strings In COMAL
Unlearning Some BASIC Ideas
Interrupt INPUT - Easy Restart
COMAL Language Notes

COMAL TODAY ISSUE #6

COMAL 0.14 Memory Map
COMAL 2.0 Memory Map
Getting Started With COMAL 2.0
COMAL Cartridge Programming Tips
Moving Up To 2.0
COMAL 2.0 From 0.14
Compare Commands - 2.0 & 0.14
Cheer Up - It Could Be Worse
Anatomy Of A 2.0 Cartridge
Draw Molecule
Shadow Letters
COMAL 2.0 Batch Files
COMAL 0.14 FIND Command
COMAL Is Compatible
Where Have All The GOTOs Gone
Color Pictures For COMAL 0.14
Substrings - Easy
Another Look At COMAL 0.14 Tokens
Goodbye GOTO

COMAL TODAY ISSUE #7

Feature: Reviews Of Every COMAL Book
COMAL In The Real World
Amazing Delete Key
Redefine Function Keys
A Simple Exercise For Non Gurus
COMAL 2.0 External Procedures
Batch Files

More ►

COMAL TODAY ISSUE #8

Easy Sprites
SAVESHAPE For COMAL 0.14
Font Sprite Maker
Converting Sprite Master Files
Sound Routines
Illusion
Random Forest
Making Stars
Fun Print
Word Game (Missing Letters)
Soundex
Names In COMAL
Function Keys
Goof-Proof Programs
Programming Batch Files
Character Codes
Using The INTERRUPT Command

COMAL TODAY ISSUE #9

Using Strings In COMAL
Beginners
Rod The Roadman
Quick Change Sprite
Tanks and the ANIMATE keyword
Function Keys
Modem Fun With COMAL 2.0
ASCII Conversion In COMAL 2.0
Display Key Values
Joy Of Trap
Detecting Drive Type
Bitmaps In COMAL
COMAL 2.0 Tokens & Internal Structure
SIZZLE - COMAL 0.14 Fastloader

COMAL Cartridge Users will especially appreciate the help the back issues beginning with COMAL TODAY #6 give in exploiting some of the more advanced features of the Cartridge. Notice the special subscriber price too! With any other purchase (including COMAL Yesterday) each issue is half the usual subscriber price - only $1.50 each. ■

RANDOM File Sampler

by Captain COMAL

During one of my trips to Denmark, one of the UniCOMAL programmers told me that Random files were easy for beginners. Personally, I prefer Sequential files, but I prepared a quick little program (for both COMAL 0.14 and 2.0) to let beginners try random files on their own. All the program does is write the names you enter to a random file. Then it prints all the names in the file. The interesting part is that it prints them in backwards order (the first name is printed last).

```
dim name$ of 20
count:=1
write'names
read'names
//
proc write'names
  open file 2,"names.ran",random 22
  repeat
    input "name: (end to stop): ": name$
    if name$="end" then
      write file 2,1: count
    else
      count:+1
      write file 2,count: name$
    endif
  until name$="end"
  close
endproc write'names
//
proc read'names
  open file 2,"names.ran",random 22
  read file 2,1: last'one
  for look:=last'one to 2 step -1 do
    read file 2,look: name$
    print name$
  endfor look
  close
endproc read'names ■
```

Metathink

— Thinking About Thinking

by Larry Phillips

The science of the principles and causes of all things existing is called metaphysics. It is a science of mind rather than that of matter. To engage in this activity, we must get outside the system we are studying. At the risk of sounding somewhat absurd, we could try to do the same with thought processes, which, if we are successful, will cause us to engage in 'metathinking', or 'thinking about thinking'. With luck, this will bring to light general principles to do with the way we humans come to our conclusions and solve problems.

Those involved with work on so-called 'expert systems' and artificial intelligence are engaging in metathinking constantly. It would seem that the only way we will ever achieve true artificial intelligence in computers will be when we can sufficiently analyze just how humans solve problems.

One of the reasons for languages such as PROLOG and LISP finding favour with AI researchers is that they tend to be more amenable to providing the sort of structures that allow a programmer to mimic the problem solving process in a way at least similar to the way he himself thinks.

Many computer languages force programmers into thinking about problems in a predetermined order. Two of the prime examples of this are assembler and FORTH; assembler because you are always working with the smallest instructions that the machine will understand, and FORTH because it will not allow you to define anything in terms that have not yet been defined. In both these examples, you must effectively program from the 'bottom-up'. This may be fine for people who actually think that way, but for the vast majority it is unsatisfactory.

COMAL, though never considered to be an AI language for many good reasons, is nonetheless a very good language in that it allows a variety of methods to be used when developing a program. I have already mentioned 'bottom up'. One other common way to program is to use a 'top-down' approach, where the programmer starts with the overall concept, then writes the necessary code to implement each of the major goals of the problem. This process continues with the blocks or modules becoming ever more detailed, until the overall goals are realized. If you are anything like me, you will program using both these techniques, and also a third. The third method could be called 'inside-out', and one of the nice things about COMAL is that it will allow all three to be used.

How many times have you looked at a problem, and thought of it only in terms of the 'nitty-gritty' details only? Or only in terms of overall goals? Not often I'll bet. With COMAL, you can write a series of statements outlining the overall goals...

```
REPEAT
 set'up
 start'game
 play'game
 announce'winner
 want'another
UNTIL no'thanks
```

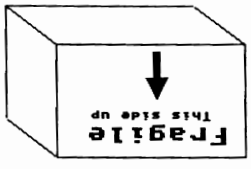
This will serve admirably to guide your thinking much as a flow chart would. On the other hand, if you happen to be thinking of a way to implement some

More ►

detailed task that you know will have to be done, you are free to write that portion of code, (as a PROC or FUNC, preferably), and to use the scan and immediate mode to test it. With COMAL's ease of MERGEing small bits of code into a larger program, you may even save the PROC or FUNC to disk for later incorporation into another program. Because of COMAL's readability, you need not even document with extensive remarks in order to later determine the way it works. Be sure though to document any portion of code that does something that would not be obvious to someone reading it for the first time.

COMAL, as implemented on the Commodore and IBM PC computers, is as much an environment as it is a language. The advantages of COMAL are evident in the ability to FIND, CHANGE, LIST by PROC or FUNC, the way it will check your syntax, and the non-destructive and informative error messages. These are not properly considered part of the language as such, but are just as important as the the actual keywords and structures for helping to guide the programmer in setting down his ideas in a logical manner.

After all, isn't that what a programming language is for? ■



FRAGILE.CRG

Packages

Spread throughout this issue of COMAL TODAY are articles describing **packages**. If you own a 2.0 cartridge, and don't know quite what to do with a package, read on. Even beginning programmers can successfully use packages.

A package is a machine language program, or to be more accurate, a sub-program. The COMAL 2.0 cartridge has 11 packages ready for you to use. Whenever you turn on the computer, the packages are available. Each package has a name, and includes some procedures. For example, the built-in FONT package has six procedures. Before you can use these procs in your program, you must tell COMAL to add them to it's list of active procedure names. Do this with the USE command. In this case, put a **USE FONT** statement at the start of your program. You may have up to 10 packages in use at once.

The packages that are on TODAY Disks are not built into the cartridge. To use them, you must bring them into memory with a LINK command:

LINK "pkg.name"

This is like LOAD, but is only used with packages. All disk-loaded packages have pkg. as the beginning of their filename, indicating they must be LINKed.

Once a package has been linked, you must issue a USE command to make it known to COMAL. After that, all procedures that are part of the package are available to you.

Once the package is LINKed in, and **USE package'name** has been issued, the COMAL program it is LINKed to can be SAVEd, and the package will be SAVEd along with it. An exception would be the rare **ROMMED** package, like META. It will not be SAVEd.■

Convert Numbers to Words
A Lesson in Recursion Using COMAL

by Mike Lawrence

Recursion is something that COMAL does well. Here is an example of how it can be useful.

The main program calls the procedure **DAC** (which stands for dollars and cents) which in turn calls **AMOUNT**, which is recursive (calls itself).

In order to write this routine, I had to think about how we say numbers, and how many words I would need to "teach" the program. We have a separate word for each number from one to twenty (20 words). Then we have thirty, forty, fifty, etc. to ninety, (7 words) then hundred, thousand, million (3 words), etc. So, to say all integer numbers from one to 999,999,999 we only need 30 different words (20+7+3).

To say numbers above 99 we say how many hundreds, or thousands, etc. (Examples: one hundred, or four thousand), but for numbers below 100 we don't do this (for example, we don't say one twenty one three for 23, we just say twenty three. From this we can see that once the routine determines that a number is 100 or above it will have to call itself to determine how many hundreds, or thousands etc., but for numbers 99 and below, it won't need to call itself. This is an important point about recursive routines; they can't keep calling themselves forever. They must eventually be able to give their "answer" without calling themselves.

Look at the listing of the procedure **AMOUNT**. After checking to see if the input X1 is zero, the routine starts scanning down through the array of values V() of the words it knows. When it finds a value that is less than or equal to its input, it then finds out how many of that value are in X1 (S) and then subtracts this term from X1. If the value is less than 100, the word corresponding to this value (W$()) is simply added to the output string X$. Otherwise the routine calls itself to translate S to words, and then tacks this word plus W$() onto the output string. Note that the procedure amount is CLOSED. This is important for many recursive routines. This routine would not work if it was not CLOSED.

A routine like this is best explained by showing an example. The technique used to produce the sample printout here could be applied to any recursive routine. Once the routine is written, an extra REF parameter is added to the parameter list (in this case: INDENT#). This variable is then incremented at the begining of the procedure and decremented by the same amount at the end. Inside the procedure, variables are printed with:

```
PRINT SPC$(indent#)
```

In this way, the information about how many times the routine has called itself is contained in the indentation of the printout in the same way that the indentation of a COMAL program shows how many structures have been placed inside each other.

In the example run below, we can see that the input to the **AMOUNT** routine was printed at the beginning of the routine, and the output at the end. Between an "In" and an "Out" we can see where the routine called itself. In this example, the routine first determined that the number was in the millions, and then called itself to find out how many millions (123). To convert 123 to a number, the routine determined that this number was in the hundreds, and then called itself to

More ►

find out how many (1). At this point the routine had 456,789 left to work with. After determining that this number is in the thousands, it called itself to find out how many (456). To do this, it then had to call itself again to find out how many hundreds are in this number (4). Now it was left only with 789. At this point it called itself once more to find out how many hundreds are in this number (7). Now it had the final answer.

By the way, if anyone wants to use this routine for writing checks, keep in mind that it is not guaranteed to always give the right answer. Very large numbers can create rounding problems which can cause the routine to make errors of a few cents.

Example run:

```
In: 123456789
How many million's ?
  In: 123
  How many hundred's ?
    In: 1
    Out: one
  Out: one hundred twenty three
How many thousand's ?
  In: 456
  How many hundred's ?
    In: 4
    Out: four
  Out: four hundred fifty six
How many hundred's ?
  In: 7
  Out: seven
Out: one hundred twenty three million,
four hundred fifty six thousand, seven
hundred eighty nine
In: 0
Out: no
```

one hundred twenty three million, four hundred fifty six thousand, seven hundred eighty nine dollars and no cents

2.0 Version

```
// delete "num'to'word"
// save   "num'to'word"
// by Mike Lawrence
//    Tucson, Arizona
PAGE
n:=30
DIM w$(n) OF 9, v(n), a$ OF 200
FOR i:=1 TO n DO
  READ w$(i)
ENDFOR i
FOR i:=1 TO n DO
  READ v(i)
ENDFOR i

REPEAT
  PRINT
  INPUT "give number: ": nu
  dac(w$(),v(),n,nu,a$)
  PRINT
  PRINT a$
UNTIL 0

DATA "one","two","three","four","five"
DATA "six","seven","eight","nine"
DATA "ten","eleven","twelve","thirteen"
DATA "fourteen","fifteen","sixteen"
DATA "seventeen","eighteen","nineteen"
DATA "twenty","thirty","forty","fifty"
DATA "sixty","seventy","eighty","ninety"
DATA "hundred","thousand","million"
DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14
DATA 15,16,17,18,19
DATA 20,30,40,50,60,70,80,90,100
DATA 1000,1000000

PROC amount(REF w$(),REF v(),n,x1,REF x$
,REF indent#) CLOSED // wrap line
  indent#:+2
  PRINT SPC$(indent#),"In:";x1
  DIM y$ OF 50
  IF x1<>0 THEN
    x$:=""
    FOR i:=n TO 1 STEP -1 DO
```

More ►

```
    IF v(i)<=x1 THEN
      s:=x1 DIV v(i)
      x1:-s*v(i)
      IF v(i)<100 THEN
        x$:=x$+w$(i)+" "
      ELSE
        PRINT SPC$(indent#),"How many
        ",w$(i),"'s ?" // wrap line
        amount(w$(),v(),n,s,y$,indent#)
        x$:=x$+y$+w$(i)+" "
        CASE v(i) OF
        WHEN 1000,1000000,1e+09
          x$:=x$+", "
        OTHERWISE
        ENDCASE
      ENDIF
    ENDIF
  ENDFOR i
ELSE
  x$:="no "
ENDIF
PRINT SPC$(indent#),"Out:";x$
indent#:-2
ENDPROC amount
//
PROC dac(REF w$(),REF v(),n,x1,REF x$) CLO
SED // wrap line
  IMPORT amount
  DIM y$ OF 50
  x$:=""
  a:=INT(x1)
  b:=INT(100*(x1-a)+.5)
  indent#:=0
  amount(w$(),v(),n,a,x$,indent#)
  amount(w$(),v(),n,b,y$,indent#)
  IF x$="one " THEN
    x$:=x$+"dollar and "
  ELSE
    x$:=x$+"dollars and "
  ENDIF
  IF y$="one " THEN
    x$:=x$+y$+"cent"
  ELSE
    x$:=x$+y$+"cents"
  ENDIF
ENDPROC dac
```

0.14 Version

```
// delete "0:num'to'word-.14"
//  by Mike Lawrence
// save   "0:num'to'word-.14"
//
n:=30
dim w$(n) of 9, v(n), a$ of 200
for i:=1 to n do
 read w$(i)
endfor i
for i:=1 to n do
 read v(i)
endfor i
//
repeat
 print
 input "give number: ": nu
 dac(w$,v,n,nu,a$)
 print
 print a$
until 0
//
data "one","two","three","four","five"
data "six","seven","eight","nine"
data "ten","eleven","twelve","thirteen"
data "fourteen","fifteen","sixteen"
data "seventeen","eighteen","nineteen"
data "twenty","thirty","forty","fifty"
data "sixty","seventy","eighty","ninety"
data "hundred","thousand","million"
data 1,2,3,4,5,6,7,8,9,10,11,12,13,14
data 15,16,17,18,19
data 20,30,40,50,60,70,80,90,100
data 1000,1000000
//
proc amount(ref w$(),ref v(),n,x1,ref x$,r
ef indent#) closed // wrap line
 indent#:+2
 print tab(indent#),"In:";x1
 dim y$ of 50
 if x1<>0 then
  x$:=""
  for i:=n to 1 step -1 do
   if v(i)<=x1 then
```

More ►

```
    s:=x1 div v(i)
    x1:-s*v(i)
    if v(i)<100 then
     x$:=x$+w$(i)+" "
    else
     print tab(indent#),"How many ",w$(i
     ),"'s ?" // wrap line
     amount(w$,v,n,s,y$,indent#)
     x$:=x$+y$+w$(i)+" "
     case v(i) of
     when 1000,1000000,1e+09
      x$:=x$+", "
     otherwise
     endcase
    endif
   endif
  endfor i
 else
  x$:="no "
 endif
 print tab(indent#),"Out:";x$
 indent#:-2
endproc amount
//
proc dac(ref w$(),ref v(),n,x1,ref x$) clo
sed // wrap line
 dim y$ of 50
 x$:=""
 a:=int(x1)
 b:=int(100*(x1-a)+.5)
 indent#:=0
 amount(w$,v,n,a,x$,indent#)
 amount(w$,v,n,b,y$,indent#)
 if x$="one " then
  x$:=x$+"dollar and "
 else
  x$:=x$+"dollars and "
 endif
 if y$="one " then
  x$:=x$+y$+"cent"
 else
  x$:=x$+y$+"cents"
 endif
endproc dac ■
```

H-P LaserJet Dump

by David Stidolph

All of the pictures in COMAL TODAY were printed on a Hewlett Packard **LaserJet** printer. Working with this printer is a dream (fast, quiet, and quality printing COMAL TODAY is printed on it). While not many of our users could afford a $3000 printer, some might be interested in the procedure to dump a Hi-res picture to it. Total dump time takes 75 seconds, and involves no machine code.

DPI refers to Dots Per Inch, and can be 300, 150, 100 or 75.

REVERSE refers to printing a negative image of the graphic screen.

```
PROC dump'laser(dpi,reverse) CLOSED
 old'map:=PEEK($c840); start:=$e000
 DIM line$ OF 42
 USE system
 setpage(0)
 SELECT OUTPUT "sp:b2400d8s1/a-/l+/t+/s2"
 PRINT ""27"*"+CHR$(116)+STR$(dpi)+"r",
 PRINT ""27"*"+CHR$(114)+"0a",
 FOR block#:=0 TO 24 DO
  FOR row#:=0 TO 7 DO
   line$:=""; base:=start+block#*320+row#
   FOR col#:=0 TO 39 DO
    addr:=base+col#*8; byte#:=PEEK(addr)
    IF reverse THEN
     byte#:=byte# BITXOR 255
    ENDIF
    line$:+CHR$(byte#)
   ENDFOR col#
   PRINT ""27"*"98"40w",line$,
  ENDFOR row#
 ENDFOR block#
 PRINT ""27"*"114"b"12"",
 SELECT OUTPUT "ds:"
 setpage(old'map)
ENDPROC dump'laser ■
```

Expert Systems in COMAL

by Michael J. Erskine

An expert system is a program which reads a data base while asking the user questions until it has identified the record containing the information which the user has been describing.

Sounds pretty fantastic, doesn't it? Well, a good one is the closest thing to artificial intelligence there is. The problem is that an expert system is designed to support a specific task. That's where the rub lies. A system designed to diagnose illnesses can do a very fine job diagnosing illnesses but can't tell you what the weather will be like in a few hours. The system is not intelligent, it simply seems that way because it has been prepared in an intelligent manner.

The game **20 questions** is a good example of how an expert system makes decisions. When people play the game the strategy becomes, "What question can I ask which will most isolate the correct answer?" In most versions of the game the person being questioned may only answer yes or no, yet it is not unusual for an astute questioner to solve the problem in less than 20 questions, even though he only receives a binary answer, a yes or a no. Why is this possible?

If there were ten million things you could think of and half were alive, all you would have to do to isolate the correct half is ask if it were alive. For either answer you would have immediatly eliminated half of the possibilities. You could then ask if it were an animal and based upon that answer eliminate another 2.5 million possibilities. This process could be continued and in a very few tries you could eliminate all but one possibility. Let's see just how many tries that would require.

In a binary system, the number of questions which must be asked to isolate a specific record (the answer) would be equal to the exponent of the power of 2 which would exceed the number of possibilities. In other words, if you had 10 million possibilities you would need to ask no more than 23 questions to isolate the item required to answer the question, "What am I thinking of?" The trick is to ask the right questions at the right time. This is what an expert system does.

In any field which requires analysis of data to produce a conclusion which is not inspired (does not require a break in logic), an expert system can be written to do the analysis, thus freeing the human to make the inspired decisions which are beyond the scope of the machine.

The easiest type of expert system to write is probably a **decision tree**, sometimes called a **rule based** program. This type of program will ask the user a question, then will either ask another question based on the answer given, or select a final answer, and present it to the user for consideration. If the user doesn't accept the answer, the system will ask him for the correct answer and for a question (RULE) which it can use to differentiate between the answer the system suggested and the answer the user expected.

In this manner the system "learns" the correct response. There are 2 programs written in COMAL 0.14 which use the decision tree structure. One by Carl Frierichs called GUESS IT, documented in COMAL TODAY #5, and TUTOR written by myself (on TODAY DISK #10). The former makes use of the fact that data statements

More ►

can be modified by a running program and all rules (questions) and answers are stored with the program. The latter uses a data base contained in a relative file which is constantly changing to reflect user input. Both programs emulate learning by association. The decision tree programs are easier to write but have the disadvantage of rigorous structure and logic which ultimately selects only one answer for presentation to the user. In other words, rule based systems cannot easily accomodate fuzzy logic or partial data. TUTOR was a quick test of rule based systems. It took about 20 hours to design and code and can easily be improved by someone having the time and inclination.

Another type of expert system uses algebraic set theory operations to select all possible answers contained in a data base, and rate, sort and present them to the user in sequence of highest probability of correctness. Set theory systems require huge files of pointers which are constantly searched as a problem is described to the program. For each change in the problem description the system must upgrade its collection of possibe answers. In such systems, the user must be able to query the data base after any change in the description of the problem. Set theory systems are much more flexible to use and considerably more difficult to code.

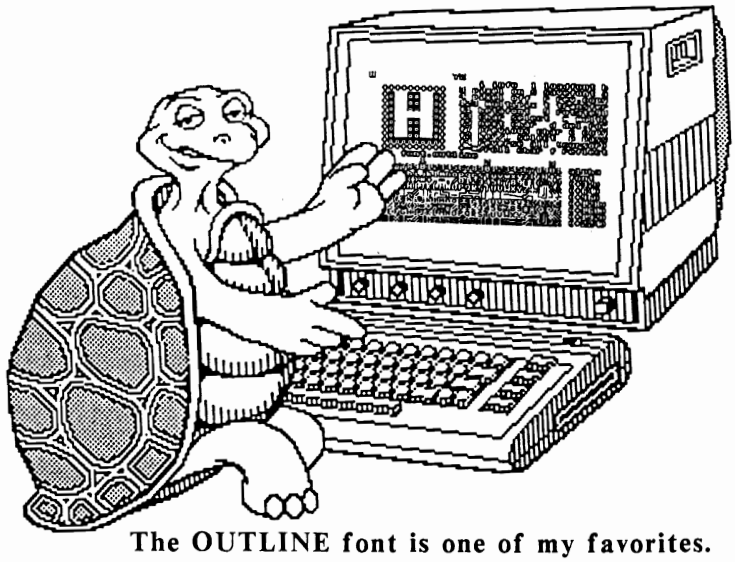
An example of this type of expert system has been written in COMAL 2.0. It maintains 2 pointer files for each question the system supports and a relative data base of up to 4000 individual records. The system can support up to 70 questions. This system probably will be made available as data bases are filled. The end user will not be able to alter the data base.

Anyone wishing more information about this system should contact: Michael Erskine, Homegrown Software, 65 East 24th Street, San Angelo, Texas, 76903, phone: 915-655-2882.

We are looking for experts who wish to join us in building a library of Expert Systems which can be read by the common reader program. In comparison with a simple decision tree program this system took about 500 hours to write and required 3 prototypes before the current version.

Currently the rage in the computer kingdom is not bigger, faster, more storage computers but smaller, faster, smarter software. COMAL allows and even encourages a programmer to write programs which are easier to alter, maintain and read. This is very important when you are writing a serious applications program. It is even more important when you are first learning to program. If COMAL wasn't your first language, it should be your next. Five years from now it may well be the only interpeted language running on any new micro. It's just that good.

Remember "Think structure!" ■



The OUTLINE font is one of my favorites.

Sorting It Out

By Dick Klingens / Dutch COMAL Users Group

Everybody who is involved with the development of computer programs will have to deal with sorting sooner or later. In this article a few sorting procedures will be examined.

There are two different ways of sorting data:
--Internal sorting, where all data is present in the memory of the computer;
--External sorting, where all data is stored in disk files.

1. Internal Sorting

Many different algorithms have been devised to do internal sorting. Many new ones increase the speed with which the data is sorted. A few simple algorithms will follow.

To explain the different sorting algorithms, we will work with an array of (Dutch) names which must be put in alphabetical order. These names are stored in an array:

```
DIM name$(0:11) OF 3
// see 1.2 for NAME$(0)
FOR i:=1 TO 11 DO
  READ name$(i)
ENDFOR i
DATA "JAN","BEN","ZUS","PIM"
DATA "WIL","KAS","RON","HAN"
DATA "LEO","ANS","TOM"
```

N.B. All examples originate from Roy Atherton's Structured Programming in COMAL.

1.1 Sorting through Selection

If we look at the array we will see that ANS will be the first item. So, we could change the position of ANS and JAN in the array. This has two advantages: we don't need any storage space except the array itself, and the part of the array that has to be examined is minimized to items 1 and 11.

The algorithm is:
1. Choose the alphabetical smallest element.
2. Exchange this element with the element with index 1 in the array.
3. Repeat this process with the 2nd, 3rd, 4th, ...10th items.

The program looks like this:

```
PROC select'sort
  FOR start:=1 TO 10 DO
    min:=start
    FOR test:=start TO 11 DO
      IF name$(test)<name$(min) THEN
        min:=test
      ENDIF
    ENDFOR test
    exchange(name$(start),name$(min))
  ENDFOR start
ENDPROC select'sort
```

The declaration of the procedure EXCHANGE:

```
PROC exchange(REF x$,REF y$) CLOSED
  DIM help$ OF LEN(x$)
  help$:=x$
  x$:=y$
  y$:=help$
ENDPROC exchange
```

In the following programs we will also use this procedure EXCHANGE, as declared.

1.2 Shuffle In

If we know that part of the file has already been sorted, we can put the items which have not been sorted yet in front of or behind the sorted part, till the whole

More ►

array has been sorted.

The algorithm is:

1. We don't need to start with the first item because we don't have to sort just one item.
2. That's why we start with the 2nd item and end with the 11th.

To show the principle, we will look at the first five already sorted items:

```
1   2   3   4   5   6   7   8   9   10   11
BEN JAN PIM WIL ZUS KAS RON HAN LEO ANS TOM
```

KAS must be placed between JAN and PIM. We will do this by comparing it to ZUS, WIL,...(to the left), and shuffling it into the array until its position in the array is correct.

The statement to compare two items could be something like:

```
NAME$(TEST) > NAME$(START)
```

Unfortunately, this would not work if we wanted to compare ANS to BEN because there is no element left of BEN and ANS really does belong there.

So, there are two points to stop comparing:

1. There is no place we can shuffle something in; we have reached the beginning of the array.
2. We shuffle in an item, we don't reach the beginning of the array.

This can be prevented from happening if we place the item that must be shuffled in into NAME$(0):

```
0   1   2   3   4   ....
KAS BEN JAN PIM WIM ....
```

The comparing problem has now been solved. The comparison can be done as follows:

```
PROC shuffle'in
  FOR start:=2 to 11 DO
    name$(0):=name$(start)
    text:=start-1
    WHILE name$(test)>name$(0) DO
      name$(test+1):=name$(test)
      test:-1
    ENDWHILE
    name$(test+1):=name$(0)
  ENDFOR start
ENDPROC shuffle'in
```

1.3 Exchanging

First the method is shown by means of an example. From

```
1   2   3   4   5   ....
JAN BEN ZUS PIM WIM ....
```

we compare the first two items. Because the order is incorrect we will exchange them:

```
1   2   3   4   5   ....
BEN JAN ZUS PIM WIM ....
```

Then, we compare items 2 and 3. We don't need to exchange them. So we compare items 3 and 4 and exchange them. This gives:

```
1   2   3   4   5   ....
BEN JAN PIM ZUS WIM ....
```

After comparing the items 10 times, we would find:

```
1   2   3   4   5   6   7   8   9   10  11
BEN JAN PIM WIM KAS RON HAN LEO ANS TOM ZUS
```

The postion of ZUS is now correct. Now we

More ►

will look at the items 1 to 10 in the same way. Then we will look at the items 1 to 9, until we have done this 10 times.

The procedure is:

```
PROC bubblesort
  FOR times:=1 TO 10 DO
    FOR start:=1 TO 11-times DO
      IF name$(start)>name$(start+1) TH
      EN // wrap line
        exchange(name$(start),name$(sta
        rt+1)) // wrap line
      ENDIF
    ENDFOR start
  ENDFOR times
ENDPROC bubblesort
```

It is obvious that we don't need to do the entire process 10 times if the array has already been sorted. We declare a Boolean variable EXCHANGED which is set to the value TRUE if an exchange is made during a search. If no exchange is made, then the sort is complete. This procedure is:

```
PROC bubblesort2
  end':=11
  REPEAT
    exchanged:=FALSE
    end':-1
    FOR start:=1 TO end' DO
      IF name$(start)>name$(start+1) TH
      EN // wrap line
        exchange(name$(start),name$(sta
        rt+1)) // wrap line
        exchanged:=TRUE
      ENDIF
    ENDFOR start
  UNTIL exchanged=FALSE
ENDPROC bubblesort2
```

But even now we can speed up things: let's look at the array until the third searching process.

```
(Times=2):
1   2   3   4   5   6   7   8   9   10  11
BEN JAN PIM KAS RON HAN LEO ANS TOM WIM ZUS

(TIMES=3):
1   2   3   4   5   6   7   8   9   10  11
BEN JAN KAS PIM HAN LEO ANS RON TOM WIM ZUS
```

The third time we stop as soon as RON has been exchanged with ANS. Start will equal 7 at that time. The items 8 to 11 have been sorted. So, START indicates the end of the unsorted part of the array. We change EXCHANGED:=TRUE into END':=START.

So now we have:

```
PROC bubblesort3
  end':=11
  REPEAT
    exchanged:=FALSE
    end':-1
    FOR start:=1 TO end' DO
      IF name$(start)>name$(start+1) TH
      EN // wrap line
        exchange(name$(start),name$(sta
        rt+1)) // wrap line
        end':=start
      ENDIF
    ENDFOR start
  UNTIL end'=0
ENDPROC bubblesort3
```

1.4 Quicksort

All sorting procedures we have examined so far are quite elementary. The way they sort is very much like the way a human would sort the array by hand. A sorting procedure which is totally different is QUICKSORT, devised by Tony Hoare. Take a look at the algorithm:

1. Choose an element from the array.
2. Use that element as a comparing operator (pivot). The array will be split into two

More ►

parts: a part with elements bigger than the pivot, and a part with elements smaller than the pivot.
3. Begin at the left and search for an element bigger than the pivot.
4. Begin at the right and find an element smaller than the pivot.
5. Exchange those two elements.
6. Repeat this until the situation as stated in 2 is true.

This is illustrated with KAS as the pivot:

```
1   2   3   4   5   6   7   8   9   10  11
        V                           V
JAN BEN ZUS PIM WIM KAS RON HAN LEO ANS TOM
            V               V
JAN BEN ANS PIM WIM KAS RON HAN LEO ZUS TOM
                V   V
JAN BEN ANS HAN WIM KAS RON PIM LEO ZUS TOM

JAN BEN ANS HAN KAS WIM RON PIM LEO ZUS TOM
```

The index which points to the left elements is stored in LP (Left Pointer), the right index is stored in RP (Right Pointer), the pivot is stored in COMP$. The search algorithm can be described as:

```
    WHILE name$(lp)<comp$ DO
      lp:+1
    ENDWHILE
    WHILE name$(rp)>comp$ DO
      rp:-1
    ENDWHILE
 "exchange the elements if nescessary"
```

The result is that the position of KAS in the array is now correct and all the elements to the left of KAS are indeed smaller than KAS, and all the elements to the right of KAS are bigger than KAS itself. The program to do this is as follows:

```
  lp:=1; rp:=1
  comp$:=name$((lp+rp) DIV 2)
  REPEAT
    WHILE name$(lp)<comp$ DO
      lp:+1
    ENDWHILE
    WHILE name$(rp)>comp$ DO
      rp:-1
    ENDWHILE
    IF lp<=rp THEN
      exchange(name$(lp),name$(rp))
      lp:+1; rp:-1
    ENDIF
  UNTIL lp>rp
```

Now we should do the same with the left and right part of the array. The program is now translated into a procedure which calls itself (recursion):

```
PROC quicksort(lend,rend)
  lp:=lend; rp:=rend
  comp$:=name$((lp+rp) DIV 2)
  REPEAT
    WHILE name$(lp)<comp$ DO
      lp:+1
    ENDWHILE
    WHILE name$(rp)>comp$ DO
      rp:-1
    ENDWHILE
    IF lp<=rp THEN
      exchange(name$(lp),name$(rp))
      lp:+1; rp:-1
    ENDIF
  UNTIL lp>rp
  IF lend<rp THEN quicksort(lend,rp)
  IF lp<rend THEN quicksort(lp,rend)
ENDPROC quicksort
```

2. External Sorting

External sorting is only necessary if there is so much data that the computer's memory fails to hold all of it at the same time.

Let's say that the data is stored in a

More ►

sequential file. Split the file in parts which can be sorted internally. If we have the same data as seen earlier, we have got to deal with a sequential file full of strings. We can now DIM:

```
DIM A$(300) OF 25
```

This occupies 300 times 27 bytes for a total of 8100 bytes, about 8K.

First a program which splits a file into a number of subfiles:

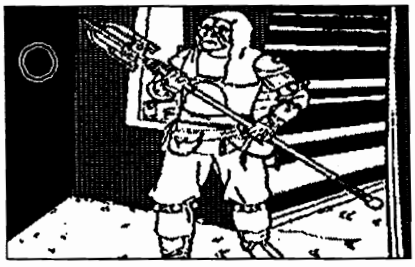
```
DIM text$ OF 25
OPEN FILE 2,"example.dat",READ
subfile:=0
REPEAT
  subfile:+1; items:=0
  OPEN FILE 3,"part"+CHR$(subfile+48),WRITE
  REPEAT
    items:+1 // number of items read
    READ FILE 2: text$
    WRITE FILE 3: text$
  UNTIL items=300 OR EOF(2)
  CLOSE FILE 3
UNTIL EOF(2)
CLOSE FILE 2
DELETE "example.dat"
PRINT "Number of subfiles =";subfile
```

Now the subfiles Part1, Part2, ...must be sorted with an internal sorting procedure and stored again in a file with the same name.

Then we must merge the subfiles into one big file. The following program assumes only two subfiles:

```
DIM one$ OF 25, two$ OF 25
in1:=2; in2:=3; outfile=4
OPEN FILE in1,"part1.dat",READ
OPEN FILE in2,"part2.dat",READ
OPEN FILE outfile,"example.dat",WRITE
READ FILE in1: one$
READ FILE in2: two$
done1:=FALSE; done2:=FALSE
REPEAT
  // Read until either file is empty
  IF one$<=two$ THEN
    WRITE FILE outfile: one$
    IF EOF(in1) THEN
      done1=true
    ELSE
      READ FILE in1: one$
    ENDIF
  ELSE
    WRITE FILE outfile: two$
    IF EOF(in2) THEN
      done2=true
    ELSE
      READ FILE in2: two$
    ENDIF
  ENDIF
UNTIL done1 OR done2
IF done1 THEN // Copy rest of 2
  WRITE FILE outfile: two$
  WHILE NOT EOF(in2) DO
    READ FILE in2: two$
    WRITE FILE outfile: two$
  ENDWHILE
ELSE
  WRITE FILE outfile: one$
  WHILE NOT EOF(in1) DO
    READ FILE in1: one$
    WRITE FILE outfile: one$
  ENDWHILE
ENDIF
CLOSE
```

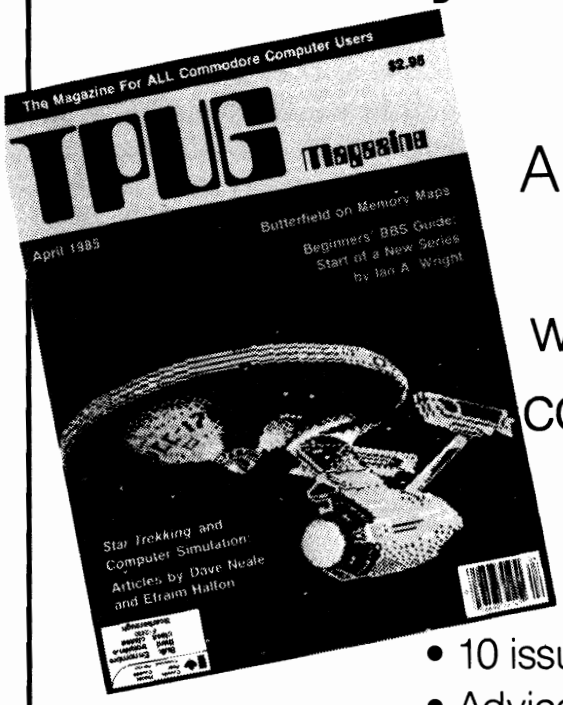
If we have more than two files we can merge them a pair at a time into one big file. ■



GUARD.CRG

TPUG

MORE THAN JUST ANOTHER COMPUTER MAGAZINE!



A membership in the world's largest computer club will provide you with:

- 10 issues of TPUG magazine
- Advice from experts like Jim Butterfield and Elizabeth Deal
- Access to our huge library of public domain software
- Access to our online help services
- All for only $25/yr.

TPUG
101 Duncan Mill
Suite CT
Don Mills, Ontario
CANADA
M3B 1Z3

JOIN US NOW!!

I want to join TPUG as an associate member.

Name _____ Home phone (____) _____

Address _____ Work phone (____) _____

City/Town _____ ☐ Cheque enclosed

Prov/State _____ ☐ Money order enclosed

Postal/Zip Code _____ Country _____ ☐ Visa ☐ MasterCard

My computer equipment is: _____ Card# _____

_____ Expiry _____

_____ Signature _____

COMAL 2.0 Sprite Quirks

by Craig Van Degrift

Although the sprite commands of COMAL 2.0 are exceedingly powerful and quite convenient to use, there are some pitfalls which can be avoided by expanding and correcting some of the information given in the Captain COMAL book 4, Cartridge Graphics and Sound.

Here are a few comments to illustrate the use of certain sprite commands which I believe need some further explanation. While working with sprites, I've found some anomalies with SPRITECOLLISION that need to be examined more closely.

After some playing around, I came to the conclusion that I could only obtain reliable results if I first waited for a jiffy clock tick (VIC chip interrupt), then did a collision test with the reset flag true and ignored its result, waited for another jiffy clock tick, and finally made a second collision test also with the reset flag set. Any subsequent need for that test result could then be satisfied by the collision test with the reset flag false as documented. I use the following function when checking for a sprite collision:

```
FUNC sprite'collision(sprite#) CLOSED
 IMPORT spritecollision
 IMPORT wait'for'tick
 wait'for'tick
 x:=spritecollision(sprite#,1) // clears
 //         old flag -- don't use result
 wait'for'tick // this is necessary to
 //               make next test valid
 RETURN spritecollision(sprite#,1)
ENDFUNC sprite'collision
```

```
PROC wait'for'tick
 time0:=TIME
 WHILE time0=TIME DO NULL
ENDPROC wait'for'tick
```

Any simpler routines would be useful, but must be very thoroughly tested!

During the course of trying to figure out these collision instructions, I tried to use the SPRITEINQ items 10 and 11 which deal with sprite collisions. Eventually, I found that these are only valid when the sprite has been stopped by the collision detection ability of the MOVESPRITE instruction.

If the appropriate bits are not set in the last parameter of the MOVESPRITE instruction, SPRITEINQ will not indicate a collision even when there is one! Also, once they test TRUE, they are rest false automatically. Even when the sprites remain in collision, these flags will not be set true again until the jiffy clock has ticked once. If you need their information in more than one test, use a variable to save the result of the first test.

Another area of confusion is the determination of whether a sprite is moving. The SPRITEINQ item 9 actually returns a countdown which hits zero only if the STOPSPRITE instruction has been given or if the MOVESPRITE instruction has successfully moved the sprite to its final destination.

If it was instead stopped by one of the collision options of the MOVESPRITE instruction, the countdown will be frozen and the sprite will be considered by the SPRITEINQ instruction to be still moving! Similar problems exist with the MOVING instruction which seems to simply return

More ►

the logical result of:

(SPRITEINQ(sprite#,9)<>0)

Even a combination function which uses SPRITEINQ items 9, 10, and 11 will fail if the final position of the MOVESPRITE command is the same point as the sprite's initial position. The sprite doesn't really move in that case but rather only "runs in place" during the countdown!

Incidentally, the 0 and 1 values for the last bit of the final MOVESPRITE parameter given in the Cartridge Graphics and Sound book are reversed. They should be 0 for **move now** and 1 for **wait for STARTSPRITES command**.

The ANIMATE command seems to work as documented as long as the sprite is moving under the control of the MOVESPRITE command. A MOVESPRITE command must be issued for the ANIMATE instruction to work. Furthermore, if the MOVESPRITE command stops its countdown because of a data or sprite collision, the animation will pause until a new MOVESPRITE command restarts the sprite.

Finally, there appears to be an actual bug in SPRITEINQ(sprite#,2) for sprites 1 though 7. A simple substitute FUNCtion which returns the lower 4 bits of a peek to location $D027+sprite# is provided:

```
FUNC sprite'color (sprite#)
 RETURN PEEK(53287+sprite#) MOD 16
ENDFUNC
```

Bestselling COMAL Books

by Captain COMAL

Three Captain COMAL books consistently sold well in the past two months: COMAL From A To Z, Cartridge Graphics and Sound, and COMAL Workbook. However, I think it is important to **notice one book** that has never made it into the top 5 best selling COMAL book list. A mere 5 books kept it from the top 5 in October. And now, in its second printing, it is typeset and spiral bound. The book is Library of Functions and Procedures compiled by Kevin Quiggle. This book and disk set gives you over 140 procedures and functions, written and tested, ready for you to merge into your COMAL 0.14 programs.

September 1985

#1 - **COMAL From A To Z**
by Borge Christensen
#2 - **Cartridge Tutorial Binder**
by Frank Bason & Leo Hojsholt
#3 - **Cartridge Graphics and Sound**
by Captain COMAL's Friends
#4 - **COMAL Workbook**
by Gordon Shigley
#5 - **COMAL Handbook**
by Len Lindsay

October 1985

#1 - **COMAL From A To Z**
by Borge Christensen
#2 - **Cartridge Tutorial Binder**
by Frank Bason & Leo Hojsholt
#3 - **Cartridge Graphics and Sound**
by Captain COMAL's Friends
#4 - **COMAL Handbook**
by Len Lindsay
#5 - **COMAL Workbook**
by Gordon Shigley
- **Beginning COMAL**
by Borge Christensen

Duplicate / Copy MSD Commands

by Colin Thompson

COMAL and the MSD Dual drive work very well together. The drive has a built-in function called DUPLICATE, which may be shortened to D (the letter dee).

Both versions of COMAL will allow you to send disk commands via the PASS keyword. We can send the DUPLICATE command to the drive like this:

```
pass "d1=0"  <press return>
```

This command will cause the drive to copy the disk in drive 0 (the LEFT drive) to the disk in drive 1 (the RIGHT drive). If you are a little unsure of this miracle, follow these instructions, step by step, and you will end up with a copy of your original disk.

1. Put the "source" disk in the left drive. This disk is the one you want to make a copy of. You may want to put a write protect tab on it in case you make a mistake (you can remove it after the duplication is done).

2. Put the "destination" disk in the right drive. Make sure the write protect tab is removed. This disk does not have to be formatted. If there are data or programs on this disk, they will be erased, forever. No known programming tricks will restore this diskette afterward.

3. Type this on the screen, exactly as shown here (don't press <return> yet):

```
pass "d1=0"
```

4. Think what would happen if you made a mistake here.

5. Check the two diskettes to make sure they are in the proper places, then close both doors.

6. Hold your breath and press the RETURN key.

Here's what will happen. For the next seventeen seconds the MSD will make 35 rapid "chunk-chunk" sounds while it formats the disk in drive 1. Then, for the next one minute and 40 seconds, the red lights will flash, both heads will move and a rhythmic "chunka...chunka..." will tell you that everything is OK.

When the red lights go out, the disk has been copied. You can check it by typing this:

```
cat "1:"  <return> // <-COMAL 2.0
cat 1     <return> // <-COMAL 0.14
```

The DUPLICATE command will not copy commercial program disks. The command is not a part of COMAL, but instead is a DOS command. COMAL is only telling the drive to perform the command by sending the command with the PASS command.

If anything goes wrong while the red lights are flashing, COMAL 2.0 will report the error to you on the screen. COMAL 0.14 will not voluntarily give you this information, so you must type this to find out what went wrong:

```
status  <press return>
```

These error messages may be interpreted by reading the MSD owner's manual. Common errors are:

```
read error
drive not ready
write error
write protected
```

More ►

If you get an error message instead of a disk copy, try it again. If you continue to get a write error, the destination disk is bad. A read error usually means the source disk is bad.

The MSD DOS has another useful command we may invoke from **COMAL 2.0** - COPY. This is not the same as DUPLICATE. (COMAL 0.14 does not have the COPY keyword).

COPY will copy a single program from the left drive to the right drive. This is really a quick and easy "single file copier". We will use the COPY keyword to tell the drive to copy a file. We can copy Program (PRG), Sequential (SEQ), and Relative (REL) files with the COPY command. Don't attempt to COPY a Random (RAN) file.

For this operation to work, we need a source disk and a FORMATTED destination disk. Once again put the source in the left drive and the destination disk in the right drive. When I do this, I like to work from the disks's directory. That way I don't make as many spelling mistakes.

For our example here, let's assume we want to copy a file called "database". First put both disks in the drive. You will need enough free blocks on the destination disk to be able to write the file there, so if you aren't sure, call up the directory like this:

```
dir "1:$$"
```

That will tell you how many blocks free that disk has. Next call up the directory of drive 0 with:

```
cat
```

Press the STOP key when you see the file "database" appear. It should look like this:

```
 27   "database"        PRG
```

Next put the cursor at the start of that line. You now have to type in the COPY keyword, and also include some information that will tell the drive where to send the file. Make the line look like this:

```
copy  "database","1:*"
```

Press return now and the file called "database" will be copied from the left drive to the right, and the filename will not change. The * (asterisk) means "use the same filename".

What if you want to copy the file and change the name at the same time? We simply replace the asterisk with the filename we want:

```
copy  "database","1:dbase"
```

If you were to leave out the "1:", the file would be copied onto the disk in drive 0 (left drive). COMAL assumes the filename following the COPY keyword will be found on the default drive unless specified otherwise.

Sometimes I want to copy all the files from one disk to another formatted disk. We couldn't use the DUPLICATE command for this because that would erase the destination disk. We simply want to add some files to another disk that already has some files on it. This could be done, one by one, file by file, but that's pretty tedious. COMAL can do it faster by using the asterisk, which really means ALL.

More ►

Once again, make sure you have enough room on the destination disk, then issue this command (COMAL 2.0 only):

```
copy "*","1:*"
```

That will send every file on drive 0 to drive 1. If by chance you've made a miscalculation, and the copying process comes to a crashing halt with a "disk full" error on the screen, do this:

```
pass "v1:"  <press return>
mount "1:"  <press return>
```

That will Validate the right drive closing the open file, and Initialize drive 1.

If all this PASSing and COPYing makes you nervous, there is a COMAL 0.14 program on USER GROUP disk #1, called SD2 COPIER. It issues the commands automatically. All you have to do is type in a filename.

More information on the PASS, COPY, and DUPLICATE commands may be found in the COMAL Handbook and the COMAL 80 Cartridge Tutorial Binder. ■

LOON.CRG

DIR in 0.14

by David Stidolph

One of the nice features of COMAL 2.0 is the DIR command that can show a disk directory from within a running program. Below is a procedure for COMAL 0.14 that can do everything the 2.0 version command can do. Much of what it can do is called PATTERN MATCHING. You can get more information on this technique in your Disk Drive Manual, or look up the DIR command in the COMAL HANDBOOK (second edition), Page 47. Here are some example calls to this procedure:

```
dir("0:*",8)          all files, drive 0
dir("1:*",8)          all files, drive 1
                      (dual drive only)
dir("0:proc.*",8)     all files which
                      begin with "proc."
dir("0:*",9)          all files drive 0,
                      device number 9
                      (second disk drive)
dir("0:*=prg",8)      all PRG type files
dir("0:???.test",8)   all files that are
                      8 characters long
                      with characters 4-8
                      being equal to
                      .test.
```

```
proc dir(name$,device) closed
 trap esc-
 for x:=1 to len(name$) do
  poke 834+x,ord(name$(x))
 endfor x
 poke 26997,x
 poke 27013,device
 poke 27110,96
 sys 26996
 poke 27110,76
 poke 26997,1
 poke 27013,8
 while esc do null
 trap esc+
endproc dir
```
■

COMAL Clinic - Tips and Techniques

COMAL 2.0- To disable the RESTORE KEY:

```
POKE $c841,1
```

And to enable it:

```
POKE $c841,0
```

COMAL 2.0- C128 owners, you can use the faster clock speed in the 64 mode. This trick would be used when your program has a lot of data to process. To switch from the one MHZ clock to the two MHZ clock, use this poke:

```
POKE $d048,3
```

This will blank the screen and all internal computer operations will execute much faster. To return to the normal speed, do this:

```
POKE $d048,0
```

This will return the textscreen and slow the system's clock speed back to one MHZ. Thanks to UniCOMAL for this tip.

COMAL 0.14 & 2.0- Would you like to make all the keys on the keyboard repeat? Use this to make them repeat:

```
POKE 650,128
```

and use this to return to normal:

```
POKE 650,0
```

You can leave COMAL 2.0 and go to BASIC with the COMAL keyword **BASIC**. You can re-enter COMAL from BASIC with this SYS:

```
sys50000
```

COMAL 0.14- You can find out the state of SETMSG by PEEKING location 4312. A result of zero means error numbers will be used. A result of one means error messages will be used.

COMAL 2.0 - If you need to find the load address of a COMAL 2.0 package, the following program may be used. It will work with any version of COMAL. The package's start address will be printed in hexidecimal format.

```
OPEN FILE 2,"pkg.filename",READ
DIM line$ OF 7
INPUT FILE 2: line$
PRINT line$(4:7)
CLOSE FILE 2
```

COMAL 0.14 -From D. Teague of Anderson, Indiana, a quick way to LIST programs to the 1520 plotter:

```
POKE 26129,6
SELECT "LP:"
LIST
```

Once you set the value at 26129 to 6 you shouldn't have to change it again while your computer remains in COMAL.

COMAL 0.14- If you don't like the start up default screen colors, you can easily modify them. This works until you turn off the computer:

```
POKE 12002,<border>
POKE 12007,<background>
```

Put these POKEs at the beginning of your program. You only have to do this once.■

My "Dream" Font Editor

by Phyrne Bacon

Two years ago, when we got our computer, tape drive and Programmer's Reference Guide, and I was just learning to program, I thought I would use BASIC to write a word processor with greek and math symbols. Why do beginning programmers try to write word processors?

I suppose I got farther than most, because I actually got a font designed. I copied the roman characters from ROM to RAM, and then used DATA statements to poke in the math and greek character designs into the reverse area. You can see the font on Font Disk #1. A few months later, we got a disk drive and a copy of the Commodore BONUS DISK. It had a font editor which I never learned to use properly, but which did give me an idea of what I wanted a font editor to do.

In September, Colin sent me a preliminary version of the COMAL Font Disk #1 with M. Bokhorst's font editor (to which Tom Kuiper had added some features). It was a fairly good editor, and would also handle multicolor characters, but it wasn't quite what I wanted in a font editor.

I like to use the cursor keys to get from character to character, so I added a display of the whole font, and cursor select.

Then I started adding more and more things: cursor wrap, reverse all, multicolor toggle, font toggle, copy, transfer, roll (up, down, right, left), rotate, reflect, delete row, erase column, set row, set column, load basic font, save as basic font, query load address, exchange fonts, display font name, prompts and so on.

I kept running out of space. To compensate, I reduced the length of the information I had added. Then I reduced the length of the procedure and variable names by changing them, listing the program, and then entering it back. And finally, I shortened all the procedure, function and variable names once again. I have gotten an amazing number of out-of-memory errors, and there were a number of things I gave up trying to add.

My thanks to David Stidolph for writing PKG.ROTATE. The machine language procedure ROTATE is very fast and just what I needed to speed up my rotations and my right/left reflection.

Kuiper added directions on how to use USE FONT and LINKFONT before running the font editor. I wanted to make this automatic, and in my procedure TEST, I was able to nest two TRAP's and do everything except press return on a line with RUN on it. If I had been in BASIC, I would have just put a carriage return in the keyboard buffer. My husband, Phil, studied the COMAL 2.0 keyboard buffer, and apparently it is more complicated than the BASIC keyboard buffer. I asked Colin how to put a carriage return in the keyboard buffer, and he suggested that I study Jesse Knight's article on batch files in memory in COMAL TODAY #7. I wrote a very short batch file in memory, and it did exactly what I wanted it to do. [See related article on the next page.] So now one can use LINKFONT from a running program.

I tried to make the FONT EDITOR as much as possible like the deluxe font editor I used to dream of back when I was working on my first font. I hope you like it. ■

Perform the Impossible

by Colin Thompson

Some of COMAL 2.0's keywords may only be used in the immediate mode - when the program has STOPped or ENDed. If you want to execute any of these keywords from a running program, the two following procs may be used. The example shown below allows you to set up the computer so that a font may be used.

It works like this. Imagine that your program needs the font screen to be established before the program begins. The keywords USE FONT and LINKFONT must be issued to allow the program to use an alternative font set. Once the program has been RUN once, the proc doesn't need to be used again, hence the line of code that checks the value of location 648. That line checks to see where the text screen is. One of the things that USE FONT does is to move the text screen to another location.

The first time the program RUNs, the text screen will be at it's normal location: 1024 (4*256). Memory location 648 will hold the value of 4, so LINK'IN'FONT will be executed. When RUN is issued, the text screen will have moved, so the proc will be skipped and your main program will begin.

Make sure that you put the line that checks where the text screen is as the very first executable line in your program.

The PERFORM'STRING proc was written by Jesse Knight and was described in an article on page 32 of COMAL TODAY #7. It functions as a "batch file from memory". The batch commands to be executed are passed as the parameter a$. The commands are built up, with a carriage return ("13"), in LINK'IN'FONT, then passed.

```
IF PEEK(648)>4 THEN link'in'font
//
PROC link'in'font CLOSED
  IMPORT perform'string
  PAGE
  DIM a$ OF 100
  a$:="use font"13""
  a$:+"linkfont"13""
  a$:+"run"13""
  perform'string(a$,TRUE)
ENDPROC link'in'font
//
PROC perform'string(task$,flag#)
  FOR x#:=1 TO LEN(task$) DO
    POKE 49151+x#,ORD(task$(x#))
  ENDFOR x#
  POKE $c866,0 //lo byte
  POKE $c867,192 //hi byte
  POKE $c865,LEN(task$) //length
  IF flag# THEN STOP
ENDPROC perform'string
```

Those of you that came to COMAL by way of BASIC may recognize this technique as an acceptable substitute for the "dynamic keyboard". COMAL 0.14 may use the dynamic keyboard, but the cartridge handles the keyboard buffer in a different manner. The buffer is "pushed onto a stack" and is not directly available to cartridge programmers without this proc.

The following proc could be used, with PERFORM'STRING, to save a copy of the current font to disk from a running program.

```
PROC write'font (name$) CLOSED
  IMPORT perform'string
  DIM a$ of 50
  a$:="use font"13"savefont("""+name$+""")
  "13"run"13""// wrap line
  perform'string(a$,TRUE)
ENDPROC write'font
```

Font Editor Instructions

[Phyrne Bacon spent 4 fun-filled weeks writing the final version of the new FONT EDITOR program. Her work exemplifies the true meaning of real-world programming. This program does everything a font program CAN do. You'll find the program on FONT DISK #1, along with 40 COMAL Fonts, 35 BASIC Fonts, several support programs and a COMAL 0.14 Font Loader. We've also included the FONT EDITOR on TODAY DISK #10 as a bonus for our subscribers.]

by Phyrne Bacon

I. GETTING STARTED

Type:

RUN "FONT EDITOR"

and wait. Since FONT EDITOR is about 71 blocks long, it takes a while to load.

The program begins in edit mode with the cursor in the upper left hand corner of the character design frame which currently displays a large representation of the lowercase letter a design.

CHARACTER DESIGN FRAME

The character design frame always displays a large 8x8 representation of the current character's design. Pixels which are on are represented by reverse periods. Pixels which are off are represented by spaces.

CURSOR SELECT FRAME

The cursor select frame always displays all 256 characters of the current font (including the letters a to z, punctuation marks, and digits 0 to 9). The display is 32 columns by 8 rows with numbers along the top and the right side.

ALPHAFONT AND BETAFONT

The Commodore computer comes with two built-in fonts which, for convenience, we will call alphafont and betafont. Alphafont has uppercase/graphics. Betafont has lowercase/uppercase. Each COMAL disk font is really both fonts. The name of the font on display will appear in the upper left corner of the cursor select frame.

CHOOSE A FONT TO EDIT (f1)

To switch from betafont to alphafont, or from alphafont to betafont, press f1.

CHOOSE A CHARACTER TO EDIT (f3)

While the cursor is in the design frame, press f3. The cursor jumps to the cursor select frame. Move the cursor over the desired character and press return. The cursor and a large representation of the selected character's design will appear in the design frame. A regular sized copy of the character will appear near the top left hand corner of the design frame. The screen code of the character will appear near the top right hand corner of the design frame.

INFORMATION (i)

There are many things one can do with the font editor. Press the i key to see two very brief pages of information (press any key to see the second page - and any key again to return to edit mode).

GETTING THE DIRECTORY (g)

To look at the disk directory without leaving the program, press g. The directory listing can be slowed by pressing the CTRL key. It can be paused and restarted by pressing the space bar.

More ►

PASS (p)

To use PASS without leaving the program, press p. Then type in the command without quotes. To delete font.fancy, type:

```
s:font.fancy
```

CONVERTING FONTS

To convert a BASIC font to a COMAL font, or a COMAL font to a BASIC font see Sections II and V.

II. LOAD, VIEW AND SAVE COMAL FONTS

LISTING THE COMAL FONTS (f8)

Press f8, to see a listing of the COMAL fonts on the disk. Only files whose names begin with "font." are shown.

LOADING A COMAL FONT (1)

Every COMAL disk font is a 17 block SEQ file whose filename begins with "font." To load a COMAL font, press 1 (for load). You will be asked for the filename. The "font." will have already been typed in for you, and you need only type in the rest of the filename. After the file is loaded, the word "comal:" and the filename will be displayed center of the screen.

VIEWING A COMAL FONT

To view a COMAL font, load it, and then press f1 a couple of times to view both its alphafont and betafont. If you see a font that you would like to try, press the STOP key, and begin typing. Use the STOP key at the end of each line of typing instead of the RETURN key. That way, you can type without having to worry about error messages or damaging the program. Type RUN to return to FONT EDITOR.

SAVING AS A COMAL FONT (s)

To save both alpha and betafont as a COMAL disk font file, press s (for save). You will be asked the filename. The "font." part will already be typed in for you, along with the name of the last font loaded. To save the font with this name, just press RETURN. To save the font with a new name, just type in a new name, and put spaces over any extra letters.

III. EDITING A CHARACTER

CURSOR

The cursor's position in the design frame is controlled by the cursor keys: down, up, right, left. The design frame has wrap; that is, if the cursor goes over the right hand edge, it appears in the left hand column; if it goes over the top edge, it appears on the bottom row.

PIXEL ON (3)

Move the cursor to where you want to turn on a pixel, and press the 3 key. The pixel will now be present in all copies of the character on screen, and will be represented by a reverse period in the character design frame.

PIXEL OFF (4)

Move the cursor to where you want to turn off a pixel and press 4. The pixel will disappear in all copies of the character on screen, and there will be a space in that position on the character design.

DELETE ROW (d)

Move the cursor to the horizontal row to be deleted, press d, and all the pixels on that row will be turned off.

More ►

ERASE COLUMN (e)

Move the cursor to the vertical column to be erased, press e, and all the pixels in that column will be turned off.

SET ROW (w)

Move the cursor to the row and press w. All pixels in that row will be turned on.

SET COLUMN (v)

Move the cursor to the column, press v, and all the pixels in that column will be turned on.

ERASE CHARACTER (CLR)

Press CLR (shift-CLR/HOME), and all the pixels will be turned off.

HOME (HOME)

The HOME key moves the cursor to the upper left hand corner of the design frame.

RETURN (RETURN)

Pressing RETURN moves the cursor one line down, and positions it on the left edge.

ROLLING DESIGN UP, DOWN, LEFT, RIGHT (+-*/)

To shift the character design up a row, press +. To shift it down a row, press -. To shift it left a column, press *. To shift it right a column, press /. These rolls are nondestructive and reversable. What goes over the top edge, appears on the bottom row, and so on.

ROTATE 90 (9)

To rotate a character 90 degrees right or left, press 9. Then choose this character by pressing t, and choose right or left by pressing r or l.

To rotate the whole font, press 9 (for rotate 90 degrees), then press w (for the whole font), and then choose right or left by pressing r or l.

REFLECT (f)

To reflect a character, press f. Then choose this character (t), and choose up/down or right/left by pressing u or r.

To reflect the whole font, press f (for reflect), then press w (for the whole font), and then choose up/down or right/left by pressing u or r.

REVERSE (r)

Press r to make a reverse copy of the character in the design frame which has, say, screen code n, at n+128 (or at n-128). The cursor is easy to see because the computer causes the character appearing at the cursor position to flicker back and forth between the screen code for that character, say n, and the reverse screen code for that character which will be n+128, or, if n>128>, n-128.

MULTICOLOR (m)

To look at the character design as it appears in multicolor mode, press the key m. This will change the whole screen to multicolor mode. Press m again to return the screen to regular mode. See Part VI.

IV. OTHER OPTIONS

EDITING THE NEXT CHARACTER (f5)

Press f5 to see the character with the next higher screen code in the design frame.

More ►

EDITING THE PREVIOUS CHARACTER (f7)

Pressing f7 will cause the character with the next lower screen code to appear in the design frame.

SELECT CHARACTER -KEY, NUMBER, CURSOR (f2)

To select a character to edit by screen code, press f2. Choose n (for number). Then press f3. You will be asked for the number of the character you wish to edit. The choice mode will remain "choose by number" until f2 is used again.

To select a character to edit by pressing a key, press f2. Choose k (for key). Then press f3. You will be asked to press the key of the desired character, and then asked if you want the reverse character. The choice mode will remain "choose by key" until f2 is used again.

To return choice mode to cursor select, press f2, and choose c (for cursor).

REVERSE ALL (CURSOR) (f4)

The cursor uses reverse copies of each character. A reverse copy of a character has a pixel off wherever the character has a pixel on, and has a pixel on wherever the character has a pixel off.

The f4 key makes a reverse copy of each character with screen code n (where 0<=n<=127 at n+128). These reverses are necessary for the cursor.

Since reverse period and reverse space are used in the character design frame, if there are no reverse characters in a font, press f4 to make the character design appear suddenly in the character design frame.

COPYING CHARACTERS (c)

If there is a character or a set of characters which you wish to copy from one place to another in, say alphafont, or from alphafont to betafont, press c. You will be shown a font and asked whether you want to copy from this font. Then you choose a beginning and ending character. Next you decide if you want to copy to this font. Finally you choose the beginning character to copy to. If you indicate you want to make the change, all the characters from the beginning character to the ending character will be copied to the font of your choice beginning with your copy-to beginning character. This can be used to copy as little as one character, and as much as a whole font.

TRANSFERRING CHARACTERS (t)

Transferring a character or characters from a COMAL disk font to a resident font is almost exactly like copying using c. However, the disk font filename must be given, and rotating back and forth between the two COMAL disk fonts and the two resident fonts is slow.

CHANGING THE FONT NAME DISPLAYED (n)

Before you begin editing a font, you may wish to change the font name on the screen. Press n and type in the new name. To make the name go away completely, press n and then RETURN.

EXCHANGING ALPHAFONT AND BETAFONT (x)

Press x to move the alphafont to the beta position, and the betafont to the alpha positon. Pressing x again will move them back as they were.

More ►

EXIT (z)

Press z to exit FONT EDITOR. To work directly upon the disk directory, scratching and renaming files, and so on, press z (or the RUN/STOP key), work on the directory, and then type RUN. This does not harm the font. But, do **not** use USE FONT, USE SYSTEM, etc. while stopped.

V. BASIC FONTS

LOADING A BASIC FONT (b)

Most BASIC font files are nine block PRG files with set. as the first part of the filename (example: SET.FANCY.A). If the filename ends with .A it is an alphafont. Betafonts end with .B. Press b (for load BASIC), and you will be asked the filename. After the BASIC font is loaded, the word "basic:", the filename and the load address will be added near the center of the screen.

If a BASIC font has a real load address, the load address will be a multiple of 8*256. If the load address is divisible by 16*256, the font is being used as an alphafont; otherwise, it is a betafont. All nine block BASIC fonts will be loaded as alphafonts.

The much rarer 17 block BASIC fonts need a multiple of 16*256 as a load address.

Once a BASIC font has been loaded into COMAL, it can be saved as either a COMAL or a BASIC font, and can be edited like any other font.

FINDING (QUERYING) A LOAD ADDRESS (q)

If you have created a font, say NEWFONT, and want to save it as a BASIC font to replace some other BASIC font, say SET.OLDFONT.A, you will need to find the load address of OLDFONT. You can find this address by putting the disk with OLDFONT in the disk drive, pressing q, then answering SET.OLDFONT.A when asked the filename. Write down the load address when it appears.

SAVING AS A BASIC FONT (a)

Press a to save a font as a BASIC font. You will be asked whether you want to save alphafont, betafont or both. Then you will be asked the load address. Remember that a single font can be saved with a load address which is a multiple of 8*256, but a double font needs a load address which is a multiple of 16*256. BASIC font load addresses are extremely tricky, so I recommend that you use an existing program to load a BASIC font (using the same load address as the usual font it uses), or that you study the Commodore 64 Programmer's Reference Guide, and pay careful attention to the parts about video bank selection, screen memory and character memory. The character memory and the screen memory must be in the same bank. This interesting information can help you understand how the BASIC fonts work, even though it is not necessary to learn it in order to use the fonts with COMAL.

VI. EDITING A MULTICOLOR CHARACTER

MULTICOLOR (m)

COMAL Multicolor font filenames carry the prefix font.mc. There are three of these fonts on FONT DISK #1.

To look at the character design of the current character as it appears in multicolor mode, press m. (Pressing m again will return the screen to regular

More ►

mode.) The cursor is now the same height, but twice as wide. In multicolor mode, it takes two pixels to specify color: 00 gives the background color, 11 gives the color specified in color memory, and 01 and 10 give background colors #1 and #2 respectively.

In multicolor character mode, a character space can be put in regular mode if the corresponding number in color memory has bit 3 (=8) equal to zero. In multicolor, most of the letters on the screen have color number eleven (3(cyan)+8) in color memory. However, the regular sized copy of the current character near the top left hand corner of the design frame has color number three (cyan), and, since it is not multicolor, it is very useful in determining which character one is working on. To see a regular size copy of the current character in multicolor, look at the cursor select frame.

BACKGROUND COLOR (4)

Move the cursor to the desired position. Set the color pair to 00 by pressing 4.

COLOR MEMORY COLOR (3)

Move the cursor to the desired position. Set the color pair to 11 by pressing 3.

BACKGROUND COLOR #2 (2)

Move the cursor to the desired position. Set the color pair to 10 by pressing 2.

BACKGROUND COLOR #1 (1)

Move the cursor to the desired position. Set the color pair to 01 by pressing 1.

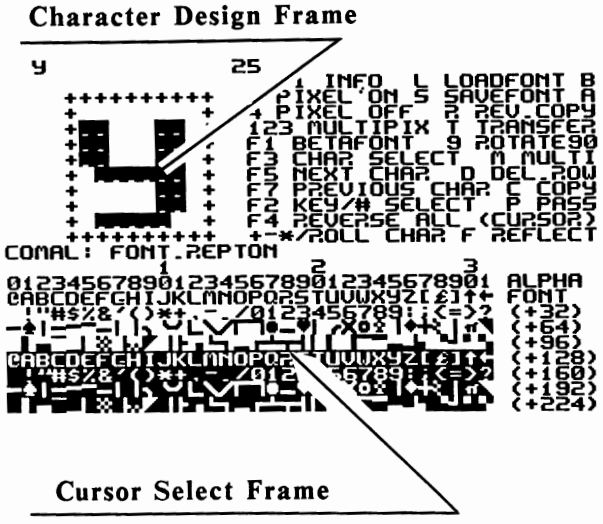
OTHER MULTICOLOR EDITING

All the other editing controls work in multicolor; however, v sets the double column to the color memory color, and w sets the row to the color memory color. To easily change a column to, say, the 01 color, press m (to exit multicolor) move to the first column of the double column, and press e (column erase). Then press m to return to multicolor.

For more information about multicolor characters, consult Chapter 3 of the Commodore 64 Programmer's Reference Guide.■

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

The Font Editor Screen



How to Use Fonts in COMAL 0.14

by David Stidolph

The Commodore 64 contains, in ROM, a set of character definitions that are used by the computer to make the letters that you see on the video screen. The character definitions are called **FONTS**. A complete font contains 4096 bytes. That's enough to make two full "sets" of characters: an Upper/Lower, and an UPPER/GRAPHICS set. Each set has 128 characters, and 128 reversed characters.

In the past, COMAL 0.14 users could not change the font to another set due to severe memory limitations. Remember that a full "double" font needs 4K of space in the computer.

[Ed. Note- No one has yet written a Font Editor for COMAL 0.14. The PROCs included in this article would form the basis for such a program. Any takers? Let us know.]

Three things must be done to use a new font. First, the data for the font must be loaded from the disk and placed in an unused area of memory. That's a problem in COMAL 0.14. There is very little unused memory, so we cheated and used the area normally reserved for sprite images and error messages.

> **RAM error messages and sprites 16 and up will not work when the new font is installed.**

Because there is not enough free RAM, only half of a full font can be used. This half-font set (2048 bytes) is called, for historical reasons, a BASIC FONT.

You can identify a BASIC font file on the disk like this: it is a 9 block PRG file, with a "SET." prefix. If the filename has a ".A" suffix, it is the UPPER/GRAPHICS set. A ".B" suffix means it is the UPPER/LOWER set.

The next step towards the installation of a font set is to move the text screen (and inform the computer where it went). Finally, we must inform the VIC II chip where to find our new font data.

Listed below are procedures that will accomplish all three deeds. These procedures can load a BASIC font, change the pointers to the new character definition, move the text screen to its new location (**set'text**), and restore the text screen to the normal default settings (**restore'screen**).

The proper sequence is:

```
dim font'name$ of 16
font'name$:="set.font'name"
loadfont(font'name$)
set'text
<...> // various commands
restore'screen
```

The process is not hard to understand, nor difficult to use. When leaving the graphics screen, however, you must now use the **set'text** procedure instead of the normal SETTEXT command.

```
proc loadfont(name$)
 if peek(850)<>169 then load'init
 for i:=1 to len(name$) do
  poke 827+i,ord(name$(i))
 endfor i
 poke 859,i-1
 sys 850
endproc loadfont
```

More ►

Fonts in COMAL 0.14 - continued

```
proc load'init
 checksum:=0
 for i:=850 to 881 do
  read num
  poke i,num
  checksum:+num
 endfor i
 if checksum<>3972 then
  print "data error!"
  stop
 endif
 //
 data 169,8,170,160,0,32,186,255
 data 169,16,162,60,160,3,32,189
 data 255,169,0,162,0,160,200,32
 data 213,255,169,8,32,195,255,96
endproc load'init
//
proc set'text
 settext
 poke 56578,(peek(56578) div 4)*4+3
 poke 56576,(peek(56576) div 4)*4
 poke 53272,18
 poke 648,196
endproc set'text
//
proc restore'screen
 poke 56578,(peek(56578) div 4)*4+3
 poke 56576,(peek(56576) div 4)*4+3
 poke 53272,22
 poke 648,4
endproc restore'screen
```

We've included a demo program on the 0.14 side of TODAY disk #10 called **VIEW'FONT/DEMO**. It will load and install a new font set for you. We have also included some extra font sets.

The FONT DISK may be used by both versions of COMAL. There are 40 double fonts and 35 single (BASIC) fonts on the disk, along with COMAL 0.14 software to load the BASIC fonts. A COMAL 2.0 Font Editor, and other support software is also included for cartridge owners. ■

Real Help for Real Beginners

by Len Lindsay

When you are just starting with computers, things seem very confusing. It becomes easy once you understand what is going on. Remember - you are not alone. All of us were beginners once.

Today, I'd like to share a couple recent inquiries we received. If you are just starting you might find yourself in a similar position.

TO LOAD or NOT TO LOAD

C. S. Wengelewski was having problems getting started. Here is what he wrote: "I tried to load COMALERRORS but it wouldn't load. Then I tried to load CALCULATOR.HRG and the computer crashed."

There is one important thing to remember when you first are starting: you can't load everything on the disk. Some things on the disk are not meant to be loaded. They are there for other reasons. COMALERRORS is a data file that COMAL 0.14 uses to get error messages to print instead of just an error number. CALCULATOR.HRG is not a program. It is a picture file. The clue that a file on the disk is not meant to be loaded is when it has a dot (.) in its name. If the file name ends with .HRG you know it is a picture file and never should try to load it. We often refer to these nonloading files as **DATA FILES**.

Look at the directory of our disks. We try to clearly separate the files that are programs (the ones you can LOAD and RUN) from the files that are not. For example, a partial directory from our AUTO RUN DEMO DISK follows (this disk was part of the $11 special):

AUTO RUN DEMO
```
"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
">---programs---<"
"menu"
"Arabesque"
...
"Towers of Hanoi"
"---------------"
">  data files  <"
">    follow    <"
">--------------<"
"abc.sprite"
"calculator.hrg"
"glady.hrg"
```

The first files at the top of the directory are the COMAL System. The first is a basic program that you use to start COMAL. The second is the COMAL 0.14 system itself. The third is the error messages used by the COMAL System. Finally, HI is the COMAL program that runs automatically when COMAL starts up.

Immediately after them notice the line:

```
">---programs---<"
```

That one line lets you know that the programs come next. Program files are the ones that you can LOAD and RUN like this:

```
LOAD "menu"
RUN
```

Or you may use one command that is the equivalent of both LOAD and RUN at one time:

```
CHAIN "menu"
```

You can see the COMALERRORS file is before the >---programs---< banner. It is

More ►

separated purposely so you will not try to LOAD it.

Finally, notice that the files at the end of the directory are preceded by the banner:

```
"---------------"
">  data files  <"
">    follow    <"
">--------------<"
```

Remember, data files are NOT programs. Do not try to LOAD them. We group them together so that we may clearly mark them as data files. Notice, CALCULATOR.HRG is among them. Read the disk directory carefully and you won't have problems trying to LOAD a file that is not meant to be loaded.

MAKE A BACKUP COPY - IT IS IMPORTANT

OK, you say. But how? The answer to this is so important that we have included a program on **TODAY DISK #10** that will backup a disk for you. Instructions for using the program are included as a separate article on page 60.

MAKE YOUR OWN COMAL STARTUP DISK

Donald L Bourdreaux wrote to ask for more details on how to create his own disk that he can use each time he wants to start up COMAL. He wants it to include the fastloader to save him loading time too (the ML.SIZZLE fastloader was explained last issue and is on both TODAY DISK #9 and #10). Here is how to make your own COMAL boot disk:

Get a blank disk. Put the disk into the drive and issue the format command:

```
From BASIC:
OPEN 15,8,15,"N0:DISKNAME,ID"
CLOSE 15

From COMAL:
PASS "N0:DISKNAME,ID"
```

Copy the COMAL system files onto the new disk. The files to copy are:

```
boot c64 comal
c64 comal 0.14
hi
ml.sizzle
comalerrors
```

The first program is a BASIC program that takes care of everything for you. Make sure you use the files from **TODAY DISK #10** since they make use of the fastloader (the COMAL 0.14 system itself has not changed since November 1983 when it was released). Make sure that the boot program is the first one on the new disk. Then you can get started with the commands:

```
load "*",8
run
```

If you don't have a program that can copy files, we have included a single file copier program on TODAY DISK #10 for your use. It is named COPYFILES.BASIC.

Once you have copied the files, the disk is ready to use. Start up COMAL with the following command:

```
load "boot*",8
    or
load "*",8
    then
run
```

You can customize the system so that it runs any COMAL program you want automatically! And it is easy too! When

More ►

COMAL starts, the first thing it does is LOAD and RUN the program named HI from the disk. You can change the HI program to any COMAL program you like. Just name the program you want to automatically start HI. Or, if you wish to keep your programs with their original names, just make a HI program that is merely one line long:

```
, 0010 chain "start"
```

Of course, you will replace "start" with the name of your program you want to be run first. Save this one line program as "hi". Now when COMAL starts up, it will load and run this HI program. Then the HI program will load and run the program with the name you chose! Automatically! ■

Change 8 to 9

by David Stidolph

If you own two disk drives you can change one of them to unit number 9 with these procedures. Connect both drives to the computer, turn on only the drive you want to be number 9, run the correct procedure for your COMAL, then turn on the other drive. This will work with both the MSD and 1541.

```
PROC change'8to9 // *** 2.0 ***
  OPEN FILE 15,"u8:/s15"
  PRINT FILE 15: "m-w"119""0""2")i",
  CLOSE FILE 15
ENDPROC change'8to9

proc change'8to9  // *** 0.14 ***
 open file 15,"",unit 8,15,read
 dim s$ of 10
 s$:="m-w"+chr$(119)+chr$(0)+chr$(2)
 print file 15: s$+")i",
 close file 15
endproc change'8to9 ■
```

Random File Size Finder

This COMAL 2.0 function can be used to find out what size is used for each record in a COMAL random file. Warning: it could take several minutes for the function to determine the size of large record size files, since it must open and close the file each time beginning with record length 1. Here is an example of how the function might be used:

```
RAN'SIZE:=RANDOM'SIZE(name$)
```

The complete function listing follows:

```
FUNC random'size(name$) CLOSED
  TRAP // see if file exists first
    OPEN FILE 82,name$,READ
    CLOSE FILE 82
    length:=0
    LOOP
      TRAP // now find RECORD length
        length:+1
        OPEN FILE 82,name$,RANDOM length
        CLOSE FILE 82
        RETURN length
      HANDLER // wrong record length
        CLOSE FILE 82
      ENDTRAP
    ENDLOOP
  HANDLER // file doesn't exist
    CLOSE FILE 82
    RETURN 0
  ENDTRAP
ENDFUNC random'size ■
```

TRAIN1.CRG

Ready, Aim, Draw

by Captain COMAL

Oh no! Not another drawing program. Yes, dear readers. We are happy to present READY AIM DRAW. While the program doesn't come close to the commercial drawing programs (or to our own GRAPHICS EDITOR system we hope to release soon) - it does show you how you can combine sprites and drawing on the graphics screen. It also shows how even small programs can do this.

The program starts by setting up things (aptly named **SETUP**). You should get used to grouping parts of your program into small modules. Then give each module a name (up to 78 characters long). From then on, whenever you want to execute the lines, just call them by name. Anyway, our program uses the SETUP module to set the screen colors, setup the sprite, print directions, and ask what speed you want to use. To setup the sprite shape it calls another module, DEFINE'SHAPE (see line 150).

Use the COMAL built in DEFINE command to define a sprite shape. That shape is setup by a 63 character string - plus a 64th character that tells whether or not it is high resolution. Our program uses DATA statements to provide the proper string characters. But notice, we don't have 63 numbers. We only use the numbers needed to set up the top corner of the sprite. That is all we need. The rest of the string is set to CHR$(0) meaning blank. To do this we watch for the End Of Data (EOD). When EOD becomes **TRUE**, we know we have read the last number, and don't read any more (just continue using the last number read - which is 0).

COMAL understands what **TRUE** and **FALSE** mean. The program uses them in line 180 to set the sprite size to be expanded. Also line 40 uses them. The program continues to AIM until **TRUE** equals **FALSE** (which is never). To stop the program just hit the STOP key!

Meanwhile, the AIM module checks the keyboard. Press a cursor key and the small cross target is moved accordingly. Press the letter D to draw a line from the last point to the point you are currently aiming at. Otherwise the routine keeps changing the color of the sprite, giving a pulsing effect. ■

```
0010 setup
0020 repeat
0030   aim
0040 until true=false //forever
0050 //
0060 proc setup
0070   black:=0; white:=1
0080   background black
0090   pencolor white
0100   repeat
0110     input "speed (1-9):": speed
0120   until speed>0 and speed<10
0130   setgraphic 0
0140   hideturtle
0150   define'shape(1)
0160   identify 1,1
0170   spritecolor 1,white
0180   spritesize 1,true,true
0190   xpos:=100; ypos:=100
0200   plottext 1,1,"crsr=move d=draw"
0210 endproc setup
0220 //
0230 proc define'shape(num) closed
0240   dim shape$ of 64
0250   for x:=1 to 64 do
0260     if not eod then read temp
0270     shape$(x):=chr$(temp)
0280   endfor x
0290   data 24,0,0,24,0,0,231,0,0
0300   data 231,0,0,24,0,0,24,0
0310   define num,shape$
0320 endproc define'shape
0330 //
0340 proc aim
0350   spritepos 1,xpos,ypos
0360   case key$ of
0370   when chr$(145) //crsr up
0380     ypos:+speed
0390   when chr$(17) //crsr down
0400     ypos:-speed
0410   when chr$(157) //crsr left
0420     xpos:-speed
0430   when chr$(29) //crsr right
0440     xpos:+speed
0450   when "d","D" //draw
0460     drawto xpos+8,ypos-5
0470   otherwise
0480     spritecolor 1,rnd(1,15)
0490   endcase
0500 endproc aim
```

'Walker' - Sprites on the March

by Captain COMAL

There is no need to be frightened of sprites. COMAL adds simple commands to easily control all 8 sprites at once. **WALKER** is a simple program that animates a sprite using 4 shapes. Try the program. Watch the **WALKER** hobble across the screen. COMAL is so fast that I had to add a pause routine to slow the WALKER down.

The whole program is merely 4 lines long. Well, actually more if you count all the procedures too. I took advantage of COMAL's modular nature, and set up some procedures to perform the several routines needed. The first one is called SETUP. As you might guess it sets up things for the program. Before setting up the sprite colors and size the SETUP procedure, calls DEFINE'IMAGES. Now, that is where the sprite shapes are determined - all four of them.

Sprite images are defined by a 63 byte string, just as in BASIC. But COMAL does all the work. DEFINE puts the images in a special reserved area of memory for you! No PEEK or POKE needed. Then IDENTIFY lets any sprite use any of the defined shapes at any time, even simultaneously. But why use a 64 byte string to DEFINE our shapes? Only 63 are for the shape itself! If the 64th byte is a CHR$(0) it tells COMAL that the sprite is Hi-Res. Otherwise the sprite will be Multi-Color (like in this program). Lines 150-170 show how easy it is to set up the colors for our sprites, as well as to choose to expand their size.

The rest of the program is a simple REPEAT loop. It continues **WALKING** until you press the Q key. WALKING keeps moving our sprite across the screen while animating the images. **SPRITEPOS** in line 430 changes its position while **IDENTIFY** in line 440 gives it an image. Then line 450 pauses a bit. Change the (99) to something smaller for a shorter pause. Remove the line altogether for some really fast walking.

Finally, notice the use of MOD in lines 420 and 440. It does a division and throws away the answer. The only thing it remembers is the remainder. Thus 9 MOD 4 is 1, the remainder of 9 divided by 4. This is a convenient way to have images cycle by fours. ■

```
0010 setup
0020 repeat
0030   walking
0040 until key$="q" //Q to Quit
0050 //
0060 proc setup
0070   blue:=14; pink:=10
0080   white:=1; black:=0
0090   define'images
0100   repeat
0110     input "speed (1-10): ": speed
0120   until speed>=1 and speed<=10
0130   background black
0140   setgraphic 0
0150   spriteback blue,pink
0160   spritecolor 1,white
0170   spritesize 1,false,false
0180   plottext 1,1,"press q to quit"
0190 endproc setup
0200 //
0210 proc define'images closed
0220   dim shape$ of 64, c$ of 1
0230   shape$(1:64):=""
0240   shape$(64):=chr$(1)//multicolor
0250   c$:=chr$(0)
0260   for x=22 to 63 do shape$(x):=c$
0270   c$:=chr$(170)
0280   for x=1 to 21 do shape$(x):=c$
0290   define 0,shape$
0300   c$:=chr$(20)
0310   for x=22 to 42 do shape$(x):=c$
0320   define 1,shape$
0330   define 3,shape$
0340   c$:=chr$(60)
0350   for x=43 to 63 do shape$(x):=c$
0360   define 2,shape$
0370 endproc define'images
0380 //
0390 proc walking
0400   for walk:=1 to 319 div speed do
0410     x:=walk*speed
0420     y:=100+walk mod 4
0430     spritepos 1,x,y
0440     identify 1,walk mod 4
0450     pause(99)
0460   endfor walk
0470 endproc walking
0480 //
0490 proc pause(delay) closed
0500   for wait:=1 to delay do null
0510 endproc pause
```

Telephone Number Database

by Captain COMAL

I'm late! I better call George. Drat! What's his last name? Klaverston. No. Calverson? Wait! Why bother? You have a computer. It's good at searching for things. This program does it all for you: creates the data file, let's you enter names into it, and best of all finds a phone number when you need it. And you don't have to remember the persons full name either!

The find routine is simple. It is really done all in one line using **IN** (see line 440). The nice thing is that **IN** looks over the entire string for a match! To find the name George Galveston you could ask for George and it would match. Or ask for Galv, ston, or just G. They all will match.

And keeping track of your disk files with COMAL is easy too. The **STATUS** command reports on the status of the disk drive (see line 290). We use it to see if opening the file was OK ("00"). The first time you run the program the data file will not be there, and the program detects this and starts it for you. COMAL watches all your files for you. Whenever you reach the end of a file it sets **EOF** (for End Of File) to **TRUE**. By using **EOF** in line 420 we make sure our program is not upset if it can't find the person requested.

Did you see the **ZONE** command in line 390? In BASIC your screen is set up with a default zone width of 10, which is nice, if you happen to need that specific setting. In COMAL, you can set the zone to be any number you wish (0 is the default for none). Line 390 sets the zone to 21, perfect for use with names up to 20 characters long (which we set in line 10).

Finally, look at the multiple choice system COMAL has. Lines 80 through 180 show the **CASE** structure. **KEY$** looks at the keyboard and returns the key you just pressed (if any). It is the basis of our **CASE**. When the key is an **E** (upper or lower case) the ENTER'NAME routine is executed. When it is **F** the program first asks for a piece of information (what to look for) and then calls the routine FIND'NAME to do the search. If none of the **WHEN** cases match, COMAL provides us with the **OTHERWISE** section. All quite readable. ■

```
0010 zone 21 // set auto tab to 21
0020 dim name$ of 20, phone$ of 12
0030 dim disk$ of 2
0040 black:=0; white:=1; yellow:=7
0050 background black
0060 repeat
0070   pencolor white
0080   print "E=enter  F=find  L=list"
0090   case key$ of
0100   when "e","E"
0110     enter'name
0120   when "f","F"
0130     input "What name?": name$
0140     find'name(name$)
0150   when "l","L"
0160     find'name("")
0170   otherwise
0180     print chr$(147) //clearscreen
0190   endcase
0200 until true=false //forever
0210 //
0220 proc enter'name
0230   input "Enter name : ": name$
0240   input "Enter phone: ": phone$
0250   if name$>"" then add'to'file
0260 endproc enter'name
0270 //
0280 proc add'to'file
0290   open file 2,"phone.dat",append
0300   disk$:=status$
0310   if disk$<>"00" then
0320     close
0330     open file 2,"phone.dat",write
0340   endif
0350   write file 2: name$,phone$
0360   close
0370 endproc add'to'file
0380 //
0390 proc find'name(search$)
0400   pencolor yellow
0410   open file 2,"phone.dat",read
0420   while not eof(2) do
0430     read file 2: name$,phone$
0440     if search$ in name$ then
0450       print name$,phone$
0460     endif
0470   endwhile
0480   close
0490   print "Hit <return> when ready"
0500   while key$<>chr$(13) do null
0510 endproc find'name

E=enter  F=Find  L=List
F
What name? COMAL
COMAL Users Group     608-222-4432
```

The Missing Letters Game

by Captain COMAL

Many people are not aware of the handy string search facility built into all versions of COMAL. This is accomplished with the keyword IN. This missing letter puzzle program demonstrates how easy this is. The program is a complete puzzle system, including routines to create the files needed as well as read them back and display the puzzle text.

The program checks the disk status after the puzzle file is opened. If the status is not "00" (for OK) the create procedure is executed. This could be modified to check only for status "62" (file not found). As listed, any error sends you to the create file routine which inputs text you type in (up to 39 letters) and writes it to the puzzle data file. It counts the entries as you go. Hit RETURN on a blank line to stop. For example, you might type:

```
Programmer's Paradise Package
```

Playing the game consists of several steps. First it reads a random number of text lines from the file. It knows exactly how many are in the file from the variable COUNT. Next it prints the text on the screen, replacing some of the letters with a dash (-). Finally, it allows the player to type in the entire text message (incorrect letters are ignored). I have included an example RUN about half way through the puzzle immediately after the program listing.

You can customize and expand this program to fit your needs. Try changing the letters the program hides with a dash (see line 400). I have it look for the vowels. Replace the "aeiou" with any other letters, or even make it a variable - just add line 385 and change line 400. For example:

```
0385 input "What letters to hide: ":hide$
0400 if text$(letter) in hide$ then
```

Also, in COMAL 0.14, remember to DIM the variable HIDE$ at the top of the program:

```
0015 dim hide$ of 26 ■
```

```
0010 dim text$ of 39, disk$ of 2
0020 open file 2,"missing.dat",read
0030 disk$:=status$; count:=0
0040 if disk$="00" then
0050   count'text
0060 else
0070   close // no data file found
0080   create'text
0090 endif
0100 play'game
0110 //
0120 proc count'text
0130   while not eof(2) do
0140     read file 2: text$
0150     count:+1
0160   endwhile
0170   close
0180 endproc count'text
0190 //
0200 proc create'text
0210   open file 2,"missing.dat",write
0220   print "input text (or blank):"
0230   repeat
0240     input text$
0250     if text$>"" then
0260       write file 2: text$
0270       count:+1
0280     endif
0290   until text$=""
0300   close
0310 endproc create'text
0320 //
0330 proc play'game
0340   open file 2,"missing.dat",read
0350   for x:=1 to rnd(1,count) do
0360     read file 2: text$
0370   endfor x
0380   close
0390   for letter:=1 to len(text$) do
0400     if text$(letter) in "aeiou" then
0410       print "-",
0420     else
0430       print text$(letter),
0440     endif
0450   endfor letter
0460   print
0470   for letter:=1 to len(text$) do
0480     while key$<>text$(letter) do
0490       print "?"+chr$(157), //left
0500     endwhile
0510     print text$(letter),
0520   endfor letter
0530 endproc play'game
```

```
Pr-gr-mm-r's P-r-d-s- P-ck-g-
Programmer's Par?
```

Design a Designer Program

by Captain COMAL

Who me? A designer? Now, an artist I'm not. But look at what I created (with a little help from COMAL). And it only took a few minutes to write the program that makes the design. Best of all, you can use the program as a starting point. **The hardest part of learning to program is starting!** I learned to program on my own years ago by looking at other peoples programs and then changing them.

Now, how would you draw a box? Draw a line and turn 90 degrees. Then do that 3 more times for all four sides:

```
SETGRAPHIC 0
FOR SIDES=1 TO 4
  FORWARD 20
  LEFT 90
ENDFOR
```

That will draw a box on your graphics screen (the first line puts you in graphics mode). Try it. Now, the fun part. Let's give those four lines we just wrote a name. Let's call them **BOX**. Just add a header line above them, and an ending line after them:

```
PROC BOX
  .....
ENDPROC
```

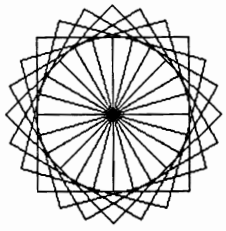
Now, once your program is run, COMAL knows what you mean if you give the command **BOX**. But always drawing a size 20 box gets boring. Let's add a parameter! When we want a box, we will tell how big at the same time:

```
BOX(30)
```

Change your **BOX** procedure to look like it does in the program listing on the side. Now you can make boxes any size you want. The procedure **DESIGN** draws a series of BOXES at different angles.

Finally, I defined my own function named **DONE**. It shows how a function can include many lines, and even ask you questions (much different than the 1 line functions of BASIC). ▣

```
0010 // save "design" // in comal
0020 repeat
0030   design
0040 until done
0050 //
0060 proc design
0070   black:=0; white:=1
0080   background black
0090   pencolor white
0100   setgraphic 0 //high res screen
0110   //pick random number & size
0120   number'boxes:=rnd(6,36)
0130   box'size:=rnd(10,60)
0140   for boxes=1 to number'boxes do
0150     box(box'size)
0160     left (360 div number'boxes)
0170   endfor boxes
0180 endproc design
0190 //
0200 proc box(length)
0210   for sides:=1 to 4 do
0220     forward length
0230     left 90
0240   endfor sides
0250 endproc box
0260 //
0270 func done
0280   plottext 8,8,"do another?"
0290   while true do //forever
0300     case key$ of
0310     when "Y","y"
0320       return false
0330     when "N","n"
0340       return true
0350     otherwise
0360       border rnd(0,15) //flash
0370     endcase
0380   endwhile
0390 endfunc done
```



Notice how easy it is to make a design using the built in turtle graphics of COMAL. Line 220 draws a line and line 230 turns the turtle to prepare for the next line drawn. Setting the screen colors is equally easy as seen in lines 70-90. Random numbers can be produced within any range (lines 120 and 130). COMAL also understands TRUE and FALSE.

How to Start a COMAL SIG

The following article first appeared in the the newsletter of the Central Coast Commodore Club, in California. In it, Tom Dipp outlines the steps he has taken to organize a COMAL SIG. We present it here so that you may see what it takes to get started. Starting a COMAL SIG is one way that you can spread the word. We also support local SIGs by providing a disk full of COMAL articles, ready to print in your newsletter. The disk is free to COMAL SIG leaders; just write and request it.

by Tom Dipp, COMAL SIG leader
Central Coast Commodore Club

Even though I couldn't attend the last meeting, thanks to Al Cooper I was able to promptly get a hold of the surveys that were filled out then. As a result, it looks like we're starting out our SIG with eight people. It'll be easier to start out working with such a small group, and build up later if the need arises. So the next thing to do is actually get the SIG off and running. To do that requires activities, a meeting place, and a minimum of organization.

To start off the activities, I thought it would be an excellent idea to get the SIG members thinking and programming in COMAL first. According to my survey, this was the most popular reason people joined the SIG. So this means a short course in COMAL and structured programming. I was thinking on the order of about 10 lessons, one lesson a week. Now I want to state for the record that I have never taught a class before, so I can only guarantee that you get what you pay for (my time is free, your course materials are not) and take your chances on the rest. However, I know quite a bit about COMAL, computers, and programming in general, and with our small group, I'm sure our 'workshops' can be fun as well as educational. And as far as the course materials, I decided to keep the cost down by using COMAL 0.14 materials.

As such, if you're in the SIG, you should already have copies of the club's COMAL and TUTORIAL disks, or be getting them soon. Other materials you'll need are the books COMAL FROM A TO Z and the COMAL WORKBOOK. These are relatively inexpensive, and normally cost about $14.00 normally for both, without shipping charges. Members would pay about $10.00 without shipping. I went ahead and made the minimum bulk purchase of 25 sets of these two books, and was able to get the price down to around $8.00 for both, including shipping. Since the COMAL SIG and myself won't need about 10 to 15 sets of these books at this time, I'll bring them to the next club meeting and sell them to anyone who wants them for $4.00 a book. The only other stuff you'll need from me for the course are xeroxed handouts.

Currently I'm looking into finding a place for our SIG to meet. According to the surveys, Monday nights are the best for evening meetings. I live in Lompoc and have a nice empty garage, but I want to find someplace closer to where most of the SIG members reside. As such, Santa Maria would be the best choice. One possibility might be the bowling alley near Bradley and Clark. If anyone has any suggestions, let me know. If not, we may have to decide on where to meet at our first meeting.

As a SIG, we'll need a little organization. As a minimum, we'll need a SIG Leader, myself for now. I'll also act as 'teacher' for now. Later if the SIG grows,

More ►

it could need a standard complement of leadership mirroring the club's itself, though I doubt it'll get that large. Also, even though we have a parent club, we may want to decentralize things somewhat and organize our own COMAL library and librarian to relieve the club librarian, especially if we want to do a lot of procedure trading.

Finally, I would like to see our SIG contribute COMAL articles to each club newsletter, and I can give a lot of help to anyone in the SIG who wants to become our SIG Editor. Don't let this scare anyone away from the SIG--we don't need a lot of organization, and the time involved in any of the positions we create will be minimal as well.

The above pretty well sums up what's going on with the SIG. As soon as we get a meeting place I'll call the members up and we'll have our first orientation meeting, which will involve such things as letting the members get to meet each other, COMAL demos, discussion of our SIG goals and logistics, and registrations and pickup of course materials for the SIG's COMAL minicourse.

I want our SIG to be a dynamic SIG that has a lot of interaction and participation among the members. This should be no problem with a small group as everyone will probably get to know each other. We might even have some parties or picnics together every now and then. I'm definitely open to suggestions, so let's make this a fun SIG.■

COMAL

Book Review
Starting with COMAL

STARTING WITH COMAL
by Ingvar Gratte
Prentice Hall International, (UK) Ltd.
published 1985, cost $15.95, 203 pages
Available from the COMAL Users Group, USA

Book review by Ian McPhedron

Starting with COMAL is for the beginning COMAL programmer. It instructs the reader in programming in COMAL-80 by guiding them step-by-step through structured programming essentials. COMAL is essentially the same, from computer to computer, so the book can be applied to any computer system which uses COMAL. Areas which are machine or implementation dependant are noted, and a space is given in the text for the reader to enter the quirks of their system.

Many examples are given to illustrate programming practice and the book includes problems and exercises to solve. Answers for many of these are given, so that the reader is not just stumbling blindly. However, other problems are left unsolved, so that the reader will learn by doing.

Ingvar Gratte is a lecturer at the University of Uppsala in Sweden. His book is a good teaching aid and will be useful for those just learning how to program or those just 'starting with COMAL', as well as for people who teach programming. The text starts with simple assignment statements, and goes through structured programming (including PROCs and FUNCs) to sequential and random disk files.

While I am in no position to compare this text with other introductory works in COMAL, I believe it to be an excellent choice for the beginner.■

How to Draw Anything in COMAL

For COMAL 0.14

A friend of mine got me started some time back when he asked for help in programming a Keno game in COMAL. Thanks to that project I got past the learning curve of COMAL and realized what a treasure I had in the language. Well, that same friend has been at it again. I suggested he try a simple project by himself: a COMAL program to draw a "Kilroy" picture. You know, a brick wall with a simple face peering over the top. He tried but I found that he really didn't have the first idea where to start. So, for those of you in the same boat, here is how you can draw Kilroy or anything else. I have no artistic talent worth mentioning so I will leave details of composition to your imagination. I want to show what the various COMAL graphics statements are, how to use them, and how to produce the simple geometric patterns from which many pictures are made.

The COMAL graphic commands are shown in the list below, grouped by function. We will see many of them in this article. You may wish to look up those we don't use. Most of them are pretty simple once you have a general idea of what is going on and I hope you have that idea by the end of the article.

SETGRAPHIC	SETTEXT	
FULLSCREEN	SPLITSCREEN	
BACKGROUND	BORDER	PENCOLOR
HIDETURTLE	SHOWTURTLE	TURTLESIZE
PENDOWN	PENUP	
DRAWTO	MOVETO	SETXY
LEFT	RIGHT	SETHEADING
BACK	FORWARD	
HOME		
CLEAR		
FRAME		
PLOT	FILL	
PLOTTEXT		

The first thing you need to do is to get into COMAL. From COMAL you must enter graphics mode:

```
SETGRAPHIC 0
```

That puts you in the high-res graphics mode. (There is also a multi-color mode, SETGRAPHIC 1, which works pretty much the same as high-res. Let's save it for another time.) At this point, you should see a blank screen with a triangular shaped turtle in the center and a couple of lines across the top of the screen in a different color. The large area is your graphic screen. The small area at the top is a text area where you can give commands. Let's try a couple of commands just for fun. Press the RETURN key after each command.

```
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
```

Did your turtle draw a box? You can clear your screen and start over with the command CLEAR. You can return the turtle to the starting position without clearing the screen by using the HOME command. Try them now.

```
HOME
CLEAR
```

We can control the turtle with the HIDETURTLE, SHOWTURTLE, and TURTLESIZE commands. Try these:

```
HIDETURTLE
SHOWTURTLE
```

More ►

```
TURTLESIZE 0
TURTLESIZE 5
TURTLESIZE 10
```

Next, let's get a good map of the graphic screen in our mind. There are 64,000 distinct points on the screen. They are arranged in a rectangle 200 points high by 320 wide. Each point is labeled by two numbers. The first number tells how far the point is from the left edge and the second tells how far from the bottom of the screen. Thus the lower left point is 0,0 because it is at the left edge and on the bottom. The top right point is 319,199. The center of the screen is called the "home" position of the turtle. It is located at point 160,99.

```
y  0,199....160,199....319,199
y  ...........................
y  ...........................
y  0,99.....160,99......319,99
y  ...........................
y  ...........................
y  0,0......160,0........319,0

   xxxxxxxxxxxxxxxxxxxxxxxxxx
```

Now that we know how our points are numbered, we can turn on any point we want using the PLOT command. PLOT x,y turns on the single point located at the specified x,y position. Let's try some and get a feel for our screen.

```
PLOT 160,200
PLOT 140,99
PLOT 140,95
PLOT 10,10
PLOT 315,195
PLOT 5,195
PLOT 317,5
```

Continue playing until you have a good feel for the screen coordinate system.

The next geometric figure is the straight line. From geometry we know that two points determine a line. We need a starting and an ending point. To draw a line on our screen, we first move to the starting point of our line then draw a line to the ending point. There are a couple ways of doing this. Here are some examples:

```
SETXY 23,99
FORWARD 51

MOVETO 25,99
DRAWTO 25,150
```

Or

```
SETXY 235,185
FORWARD 36

MOVETO 235,180
DRAWTO 270,180
```

How about lines at angles? Sure!

```
SETXY 170,150
SETHEADING 135
FORWARD 75

MOVETO 160,150
DRAWTO 210,100
```

Try some of your own. When you are done, clear the screen with the CLEAR command. Note that the SETHEADING command turns the turtle to the specified angle where zero degrees is straight up, 90 degrees is right, etc. Some projects will be more easily drawn with MOVETO and DRAWTO while others will do better with SETXY, SETHEADING, and FORWARD type commands.

Now let's try for some polygons starting with a triangle. We move to the first point, draw to the second, draw to the

More ►

third, then draw to the first again.

```
MOVETO 140,99
DRAWTO 160,129
DRAWTO 180,99
DRAWTO 140,99
```

I'll leave other straight-lined figures for you to work out on your own. You know how to put any kind of a straight line on the screen at any place you wish so you should really have no problems.

Since we are going to get into more complicated figures, its time to change our method of operation. Instead of typing in each command and watching it execute, we will write a program containing the commands. When it is just the way we want it, we will RUN it and have our picture drawn for us. This has several advantages. We can save our program for repeated use. We can make use of loops and other program constructs that are not available as direct commands. When we make a goof, we need only correct the program. We don't have to type all the other commands over, too.

Let's get out of graphics mode. You can give the command SETTEXT or just hit the F1 key. If you type SETGRAPHIC without a number, or hit the F3 key you will be back in split screen graphics mode and can review your picture. F5 puts you in fullscreen graphics mode. You can also change modes with the SPLITSCREEN and FULLSCREEN commands but the function keys are simpler to use. Try playing with the them now, then return to text mode (F1).

Since we start our program from text mode, the first thing it needs to do is to enter graphics mode. We will not be giving commands in graphics mode (our program does that for us) so let's use the fullscreen graphics. Finally let's hide the turtle so we can see only the points and lines we've drawn. Our program will begin like this:

```
10  // Comment to identify our program.
20  setgraphic 0
30  fullscreen
40  hideturtle
```

I haven't said anything about colors yet but let's set our background and border colors. We may need to look up the color numbers in the Commodore Programmer's Reference Guide. COMAL uses the same color numbers used by BASIC. Black is 0, White is 1, Red is 2, Green is 5, Blue is 6, etc.

```
50 background 1
60 border 5
70 clear
```

The CLEAR command will erase the graphics screen and set it all to the color we last specified with the BACKGROUND command. We can also set the color of the lines we want drawn. We do this with the PENCOLOR statement.

```
80 pencolor 6
```

We're all set now, but what shall we do? Let's draw the square we drew earlier in the command mode.

```
90  forward 50
100 right 90
110 forward 50
120 right 90
130 forward 50
140 right 90
150 forward 50
```

Now run it. Because we are in program mode, we can make use of loop and

More ►

conditional structures. FOR-ENDFOR loops are particularly useful for polygons. Let's improve the square program with a FOR loop.

```
0010  // improved square program
0020 setgraphic 0
0030 fullscreen
0040 hideturtle
0050 background 1
0060 border 5
0070 clear
0080 pencolor 6
0090 for side = 1 to 4
0100  forward 50
0110  right 90
0120 endfor side
```

With a simple change, we can make this program print any polygon. Add line 15 and replace lines 90 thru 120 with this:

```
0015 input "how many sides? ":number

0090 //
0100 for sides= 1 to number do
0110  forward 10
0120  right 360/number
0130 endfor sides
```

Try some programs of your own. You might also want to use the LEFT and BACK commands. They work like RIGHT and FORWARD but go in the other direction.

In order to draw curved lines, we cheat. The COMAL statements necessary have already been written for us and are found in the Library of Functions and Procedures book/disk set. It includes procedures that will draw circles and arcs for us. All we have to do is merge the procedures from the library disk into our program and call them as needed. The book explains how to call each routine.

To merge a library procedure or function with your program you write your program making sure that you do not use any line numbers greater than 8999. The library routines start at 9000. When you are ready, you ENTER the library routine. This will merge it from disk and put it at the end of your program. Remember that ENTER only works with SEQ files (as created by LISTing a program section to disk)!

The library disk is set up so ENTER will work. After the library routine is entered, renumber your program to make room for any additional routines you may want to merge:

```
RENUM
```

If you forget to renumber, the next routine you ENTER will overlay all or part of the first one and you will have a real mess.

It's time to start drawing Kilroy. Kilroy consists of a wall of bricks (lets keep it simple with only three rows of bricks) with Kilroy's head peeking over the wall. For extra credit, you ambitious ones can add Kilroy's fingers and nose or extend the picture in other ways. For now we need to draw two objects - a brick wall, and a head. The brick wall can be seen in several ways:

1. A lot of distinct lines
2. A box with two horizontal lines and a bunch of vertical ones
3. three rows of bricks.

The third alternative is probably the best because it is the most general description. We can easily draw a brick. It's just a variation of our square program. Next we can learn to draw a row of bricks and finally, we can draw three

More ►

such rows. By abstracting the program this way, we will later be able to change the size of the brick, the number of bricks in a row, the number of rows, etc. With the other two approaches these changes would be very difficult.

In fact, to keep things very modular, we will put each of our abstractions into its own procedure. Let's write the BRICK PROC:

```
310 proc brick(high,wide) closed
320  for side:=1 to 2 do
330   forward high
340   right 90
350   forward wide
360   right 90
370  endfor side
380  //
390  left 90
400  back wide
410  right 90
420 endproc brick
```

Notice that we had to use separate FORWARD commands for the two adjacent sides since they may not be the same length. The LEFT, BACK, RIGHT statements at the end of the proc move the turtle to the right end of the brick in position to draw an adjacent brick. This will make our procedure to draw a row of bricks very simple:

```
440 proc row'of'bricks(count,high,wide) cl
    osed // wrap line
450  for i:=1 to count do
460   brick(high,wide)
470  endfor i
480 endproc row'of'bricks
```

There is only one small problem. Succeeding rows of bricks alternate a short and a long brick at the start of the row. We can make a second proc for the other row of bricks:

```
500 proc other'row'of'bricks(count,high,wi
    de) closed // wrap line
510  short:=int(wide/2)
520  brick(high,short)
530  //
540  for i:=1 to count-1 do
550   brick(high,wide)
560  endfor i
570  //
580  short:=wide-short
590  brick(high,short)
600 endproc other'row'of'bricks
```

We had to be careful in our computations because we don't want to end up with one row of bricks shorter than another. We are now ready for our wall procedure:

```
620 proc brick'wall(x,y,rows,count,high,wi
     de) closed // wrap line
630  moveto x,y
640  case (rows mod 2) of
650  when 0
660   for i:=1 to rows/2 do
670    row'of'bricks(count,high,wide)
680    moveto x,y-(2*i*high)
690    other'row'of'bricks(count,high,wide)
700    moveto x,y-((2*i+1)*high)
710   endfor i
720  when 1
730   row'of'bricks(count,high,wide)
740   moveto x,y-high
750   for i:=1 to (rows-1)/2 do
760    other'row'of'bricks(count,high,wide)
770    moveto x,y-(2*i*high)
780    row'of'bricks(count,high,wide)
790    moveto x,y-((2*i+1)*high)
800   endfor i
810  endcase
820 endproc brick'wall
```

Again, we had to be careful because we might call for either an even or odd number of bricks. Note that we must also be sure to set our position before drawing each row of bricks.

More ►

Our main program is very much like it used to be:

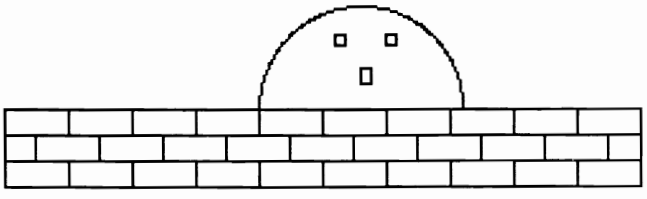
```
100 // for comal 0.14 - kilroy was here
110 // delete "0:c14.kilroy"
120 // save   "0:c14.kilroy"
130 setgraphic 0
140 fullscreen
150 hideturtle
160 background 1
170 border 5
180 clear
190 pencolor 0
200 brick'wall(20,80,3,10,10,25)
210 arc(160,90,40,0,180) // head
```

Finally we start tinkering and add the bells and whistles. We add eyes and a nose as well as a title for the picture. Here is the final version as far as I have taken it. Now it's your turn to experiment both with this picture and others of your own creation. Good luck and have fun!

```
0100 // for comal 0.14 - kilroy was here
0110 // delete "0:kilroy"
0120 //   save "0:kilroy"
0130 setgraphic 0
0140 fullscreen
0150 hideturtle
0160 background 1
0170 border 5
0180 clear
0190 pencolor 0
0200 brick'wall(20,80,3,10,10,25)
0210 arc(160,90,40,0,180) // head
0220 pendown
0230 moveto 150,115
0240 brick(4,4) // eye
0250 moveto 170,115
0260 brick(4,4) // eye
0270 moveto 160,100
0280 brick(6,4) // mouth
0290 plottext 100,10,"kilroy was here"
0300 //
0310 proc brick(high,wide) closed
0320  for side:=1 to 2 do
0330   forward high
0340   right 90
0350   forward wide
0360   right 90
0370  endfor side
0380  //
0390  left 90
0400  back wide
0410  right 90
0420 endproc brick
0430 //
0440 proc row'of'bricks(count,high,wide) c
     losed // wrap line
0450  for i:=1 to count do
0460   brick(high,wide)
0470  endfor i
0480 endproc row'of'bricks
0490 //
0500 proc other'row'of'bricks(count,high,w
     ide) closed // wrap line
0510  short:=int(wide/2)
0520  brick(high,short)
0530  //
0540  for i:=1 to count-1 do
0550   brick(high,wide)
0560  endfor i
0570  //
0580  short:=wide-short
0590  brick(high,short)
0600 endproc other'row'of'bricks
0610 //
0620 proc brick'wall(x,y,rows,count,high,w
     ide) closed // wrap line
0630  moveto x,y
0640  case (rows mod 2) of
0650  when 0
0660   for i:=1 to rows/2 do
0670    row'of'bricks(count,high,wide)
0680    moveto x,y-(2*i*high)
0690    other'row'of'bricks(count,high,wid
        e) // wrap line
0700    moveto x,y-((2*i+1)*high)
0710   endfor i
0720  when 1
```

More ►

```
0730    row'of'bricks(count,high,wide)
0740    moveto x,y-high
0750    for i:=1 to (rows-1)/2 do
0760      other'row'of'bricks(count,high,wid
          e) // wrap line
0770      moveto x,y-(2*i*high)
0780      row'of'bricks(count,high,wide)
0790      moveto x,y-((2*i+1)*high)
0800    endfor i
0810  endcase
0820 endproc brick'wall
0830 //
0840 proc arc(x,y,r,sa,ca) closed
0850  moveto x,y
0860  setheading 90-sa
0870  penup
0880  forward r
0890  left 90
0900  pendown
0910  arcl(r,ca)
0920  penup
0930 endproc arc
0940 //
0950 proc arcl(r,ca) closed
0960  for i:=1 to ca/10 do
0970    left 5
0980    forward r*3.14159/18
0990    left 5
1000  endfor i
1010 endproc arcl ■
```



kilroy was here

Bitmap — A New Package

by David Stidolph

On the 2.0 side of TODAY DISK 10 there is a package called pkg.bitmap. Its purpose is to copy the contents of the graphics screen into a string, and copy the string back to the screen.

To use the package:

```
link "pkg.bitmap"
```

To use the commands in the package you must issue the USE command:

```
use bitmap
```

This package contains two commands:

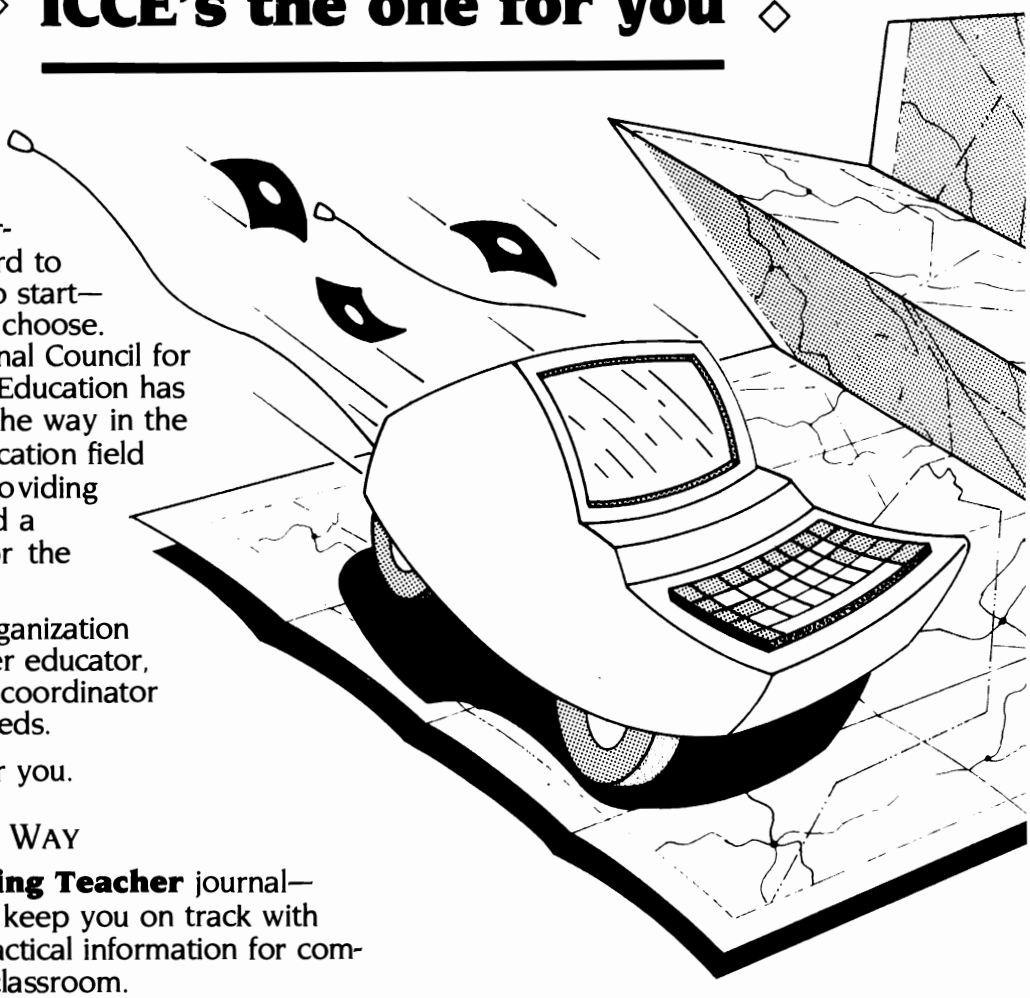
```
get'bitmap(string$) // screen to string
put'bitmap(string$) // string to screen
```

Both commands need the same parameter; an 8,000 byte string. The following short program shows how the procedures are used. The program will draw random lines on the graphics screen, save the bitmap, clear the screen, and restore the bitmap. One use of this package would be in a graphics editor for non-lethal FILLs and drawing.

```
DIM map$ of 8000
USE bitmap // activate package
USE graphics
graphicscreen(0)
// draw on graphics screen
FOR x:=1 TO 99 DO
  drawto(RND(0,319),RND(0,199))
ENDFOR x
get'bitmap(map$) // save bitmap
clearscreen // clear graphics screen
put'bitmap(map$) // restore bitmap
END "Bitmap restored!" ■
```

Take Off With Us

◇ ICCE's the one for you ◇



With today's profusion of computer information, it's hard to know where to start—which road to choose. The International Council for Computers in Education has been guiding the way in the computer education field since 1979, providing leadership and a ground plan for the future.

It's the one organization every computer educator, administrator, coordinator or librarian needs.

It's the one for you.

Guiding the Way

The Computing Teacher journal—guaranteed to keep you on track with up-to-date, practical information for computers in the classroom.

Special Interest Groups—share information to help your special interest area grow. SIGs include computer coordinators, teacher educators, administrators and special educators, and are planned for advanced placement in computer science, community colleges and videodisc users. The quarterly **SIG Bulletin** serves as a forum for SIG information.

Booklets and Monographs point to additional information on specific topics. **ICCE Packets** provide you with teacher training materials. Members receive a 10% discount on all three.

ICCE Committees address a variety of ethical and practical issues important to you as a computer-using educator.

ICCE participates in computer education conferences throughout the world, supporting our state- and region-wide member organizations.

INTERNATIONAL COUNCIL FOR COMPUTERS IN EDUCATION ICCE

Join the One for You.

Making Decisions with IF-THEN-ELSE

by Colin Thompson

One of the building blocks of the COMAL language is the IF-THEN-ELSE structure. Nearly every COMAL program uses these keywords to make decisions. Once you have mastered IF-THEN-ELSE and FOR-ENDFOR, you are well on your way to mastering COMAL.

Most programs make decisions. All decisions the computer makes are boiled down to this: something is TRUE or else it is FALSE. There are no grey areas in the computer's evaluation of your program. When asked to make a decision, the expression is always reduced to either TRUE or FALSE. Once a decision has been reached, the computer must take an action, based on your instructions. This action might be a BRANCH. When a branch is made, the program flow jumps to another part of the program and continues from there.

The simplest decision making statement is the IF-THEN. When a yes/no decision is to be made, we write a COMAL sentence like:

```
IF score=100 THEN game'over=TRUE
```

When COMAL sees this line it 'evaluates' the first 'expression', <SCORE=100>. COMAL looks up the present VALUE of SCORE and compares it to 100. It actually subtracts the two. If the result of the subtraction equals zero, then COMAL continues on to perform the statement following THEN. In this case, COMAL would make the value of the variable **GAME'OVER** equal to TRUE. In COMAL, FALSE is equal to 0, and 1 is used as the value for TRUE. If score was not equal to 100, COMAL ignores the expression following THEN and continues to the next largest line number.

Our small example here does not show how BRANCHING takes place. Regardless of the value of **score**, the program continues on the next line. The example seems to be determining if the game is over, or not. When GAME'OVER is finally assigned to be TRUE, another part of the program will take the appropriate action.

Let's rewrite the example to show how COMAL can make a decision and then take several consecutive actions, based on the lone decision. We will use the IF statement to make the initial decision, and then present COMAL with a list of statements to carry out, following the THEN:

```
IF score=100 THEN
  game'over=TRUE
  show'scores
  STOP
ENDIF
```

COMAL evaluates the expression <score=100> and makes a decision. If SCORE is equal to 100, COMAL looks at the line following THEN, and executes it. If SCORE is not equal to 100, COMAL jumps to the line following the ENDIF statement and continues from there. This is a branch.

The first example is called a **one line IF-THEN statement**. It allows only one action to be taken. The second example allows us the freedom to carry out several actions, based on a single decision. COMAL lets us to place any number of instruction lines between the IF and ENDIF.

In our second example, there are three distinct kinds of actions taken. First, the variable **GAME'OVER** is assigned the value of TRUE, indicating the game is over.

Next, the PROCedure SHOW'SCORES is called. This forces COMAL to branch to another

More ►

part of the program and carry out the instructions found in SHOW'SCORES. When completed, COMAL will branch back to the line with the STOP statement and will stop the program.

The procedure SHOW'SCORES might look like this:

```
PROC show'scores
  PRINT CHR$(19), //home cursor
  PRINT "your score"; score
  IF game'over=TRUE THEN
    IF score=100 THEN
      PRINT "you win !!"
    ELSE
      PRINT "I win !!"
    ENDIF
  ENDIF
ENDPROC show'scores
```

This PROC uses **nested** IF statements, one IF STRUCTURE within another. If you look closely, you'll see a new decision making statement: ELSE.

Let's look this PROC over closely to see what makes it tick. First, you must understand that this PROC may be called from more than one place in the program. It's primary purpose is to display the player's score at, say the end of each round, or whenever the player's starship has been blasted into it's component atoms.

The player's score is printed on the top line of the screen and then a decision is made: "Is the game over?" If GAME'OVER is FALSE, indicating the game is still in progress, COMAL jumps to the ENDIF, then the ENDPROC, and returns to where it was called. No action was taken except to paint the score on the screen.

If, however, GAME'OVER was TRUE, another decision must be made on the following line: "Who won?" Either the player won, or the computer won. We know the player won if the score equals 100. If the player won, the next line is executed, and "you won" is painted on the screen. Program flow jumps to the ENDIF, then the next ENDIF and finally the ENDPROC.

If SCORE was not equal to 100, then the computer must have won. That also implies that this PROC was called from another place in the program, and not from our example. In this case, COMAL skips to the ELSE statement and then starts executing the line following it. IF-THEN-ELSE.

"If the score is 100 THEN print 'you won' ELSE print 'I won'".

With the IF-THEN, we can take one of two actions: Either do something, or not. ELSE gives us a second option: We can do either the first or the second.

What if you needed to do one of FOUR options? Or SEVEN options? COMAL has several ways to accomplish this task. Multiple branching decisions are usually done with the CASE statement, and sometimes with the ELIF.

In any case, the decision is up to you, the programmer. ■

GUARDS.CRG

How to Make a Backup Disk

by Captain COMAL

Don't use the disks we send you! As a matter of fact, don't use the disks that any company sends you. First, backup the disk. Then store the original disk in a safe place. Finally, use the copy of the disk. "Why bother?" you may ask. Well, if for any reason the copy you use is ruined, simply get out the original disk and make yourself another copy.

OK, there are more reasons to make those copies. The main reason is compatibility. There are at least three variations on the popular Commodore disk format: 1541, 4040, and MSD. They each can read all three format types, thus they are called READ compatible. However, they are NOT WRITE compatible. If you use your 1541 drive to write on a disk that was created with a 4040 drive you are looking for trouble. Problems will not arise right away. But one day - ZAP. You will have missing files! Avoid the heartache. Never use disks sent to you by someone else. The odds are that they use MSD or 4040 drives to make their disks (that is what we do).

Many of our disks come to you with programs on BOTH sides. Since we only use premium disks specifically designed for this, it shouldn't present a problem. However, using the disk constantly from side to side is not recommended. Copy both sides of these disks.

A final reason for copying the disks is that of disk drive head alignment. Your alignment may be just slightly misaligned. That will not cause problems as long as you use disks written by your own drive.

OK, enough lectures. On with the course. Just how do you make a backup copy of our disk? If you are one of the lucky ones with an MSD Dual Drive it is easy. Just place the original disk into drive 0 (on the left) and a blank disk into drive 1 (on the right). Then issue this command:

```
PASS "D1=0"
```

If you have only a single drive, you still can backup our disks. Since our disks are NOT copy protected, any disk backup program will work. Use the backup program you already have and just follow the instructions that came with it. If you do not have a disk backup program, we are providing one for you on **TODAY DISK #10**. Since maximum memory is needed to backup a disk, the program supplied is a BASIC program. Do not try to load it into COMAL. Now, here is how to use it:

1) Remove any disk from your disk drive.

2) Turn off the computer system.

3) Turn on the computer system.

4) Using unshifted letters type in this command followed by the RETURN key:
   load "backupdisk.basic",8

5) Type in this line followed by RETURN:
   run

6) The screen clears and you are asked:
   disk name ?

7) Before replying, remove the original disk from the drive and insert a blank disk. The program erases and formats the disk for use as your backup.

8) Type in a name for the disk (up to 16 characters).

9) Next you are asked:
   unique disk id ?

More ►

Every disk has its own ID. This ID can be any two characters. No two disks should ever have the same disk ID. It is hard to emphasize this enough. The Captain COMAL Gets Organized disk management system will help you keep track of your disk ID's (even print a chart of those you have already used). For now, type in two characters that you are SURE are not used elsewhere in your disk library. Try something like [[since it is unusual.

10) Instructions now appear on the screen:
insert source disk, press SPACE

11) Remove the new disk from the drive and label it with the disk name and ID. Insert the original disk, also called the source disk. After doing this, hit the SPACE bar.

12) Information about the disk will scroll by. It is not important to read it. It ends by estimating about how long in minutes and seconds that the copy may take. The program counts and reads blocks from the source disk. The count is flashed on the bottom screen line:
reading block #---

13) After 124 blocks are read, more instructions are shown on the screen:
insert destination disk, press SPACE

14) Take out the original source disk. Put the new destination disk into the drive. After you do this hit the SPACE bar.

15) As blocks are written to the disk, the count is shown:
writing buffer #---

16) After all the blocks are written you will see the message:
insert source disk, press SPACE

17) Take out the new destination disk. Put the original source disk back into the drive. After you do this, hit the SPACE bar.

18) Now the familiar message appears again, flashing through the next set of blocks, 1 through 124 (yes, the numbers are 1-124 again, but this is a different set of 124 blocks):
reading block #---

19) After the next 124 blocks are read you'll see the message:
insert destination disk, press SPACE

20) You may now go back to step 14 and continue the cycle of swapping disks back and forth as instructed by the program. BE PATIENT! It may take up to 12 disk swaps. The last pass of reading blocks from the original source disk will probably not go all the way up to 124. It will probably end at some other number. That is your sign that this is the final swap. Put in the destination disk as instructed. After it writes the final blocks you are done. The program then lists the directory of the newly created disk.■

The following two functions are for use with the picture compression system on page 68.

```
FUNC loadcompact(name$) CLOSED
  USE compactor
  TRAP
    OPEN FILE 147,name$,READ
    load'compact(147)
    CLOSE FILE 147
    RETURN 0
  HANDLER // return error code
    CLOSE FILE 147
    RETURN ERR
  ENDTRAP
ENDFUNC loadcompact
```

```
FUNC savecompact(name$) CLOSED
  USE compactor
  TRAP
    OPEN FILE 147,name$,WRITE
    save'compact(147)
    CLOSE FILE 147
    RETURN 0
  HANDLER // return error code
    CLOSE FILE 147
    RETURN ERR
  ENDTRAP
ENDFUNC savecompact
```

EXEQ — A New Package

by Ian MacPhedran

COMAL 2.0 for the Commodore 64 has many excellent programming features, but two of the most significant ones are the use of named procedures, and the ability to use machine language routines as procedures (packages). However, these two capabilities have not been combined until now.

That is, a programmer could not write a machine code PROC (package) which could call (EXECecute) another PROC written either in COMAL or in machine code. The ability to do this would be a powerful addition to COMAL programming. This article will describe how to perform this trick.

Before continuing, I must thank Jesse Knight Jr. for his assistance with this project. Both his book, COMAL 2.0 Packages, and his personal advice have been of great help. Those programmers wishing to fully use the power of COMAL (including the topic discussed here) should have a copy of his book.

The authors of COMAL 2.0, UniComal, were extremely thoughtful when they set up the COMAL/machine language interface. They also set up a jump table for important routines, and documented them. Within this jump table is a vector called EXCUTE (at $CA36) which will execute a tokenized COMAL line stored in a buffer called CDBUF ($C661 to $C75F). This allows a programmer to use COMAL to a limited extent. However, a call to this routine will stop a running program. This limitation can be overcome however by using the following algorithm.

1; Load tokenized line into CDBUF
2; Save important locations as follows:
   a; contents of LNLEN ($1F)
   b; contents of PRGPNT ($31-$32)
   c; contents of CODPNT ($33)
   d; contents of CSTAT ($C845)
3; Call EXCUTE (JSR $CA36)
4; Restore previous values to above locations.

This may seem to be fairly simple, however, there are a few key points to notice here. EXCUTE destroys the contents of the first four locations mentioned above when executing CDBUF, thus they must be saved, or the COMAL interpreter will not know where to return within the program. The fifth location, CSTAT, contains a code which tells COMAL what mode it is in (i.e. running a program, in immediate mode, etc.). This location is also written to by EXCUTE, fooling COMAL into believing that it is in the immediate mode (the editor mode). This must be restored to its original value so that a running program will not stop. (This will also return COMAL to the immediate mode, if that is where it was called from.)

Also, the tokenized line must end with the two bytes $FF $33, which act as a STOP command. These are necessary to stop EXCUTE and return to the calling machine language routine.

The following is some information on tokenizing EXEC PROC lines.

For a PROC with no parameters
(e.g. EXEC a)
the format of the tokenized line is as follows:

More ►

```
$00 $00 $08        ;line header
$81 <off >off      ;EXEC PROC
    ;off is the offset of the PROC name
    ;in the name table
$FF $33            ;line trailer
```

For a PROC with parameters
(e.g. EXEC a(c,f,56,"0"))
the format is slightly different:

```
$00 $00 length     ;line header
    ;length is the number of bytes in
    ;the line
$1A <off >off      ;EXEC PROC
parm 1             ;parameter
parm 2             ;specifications
. . .              ;see below
parm n             ;last parameter
$FF $33            ;line trailer
```

The parameter specification statements are as follows:

```
    real variable    : r
$07 <off >off $10

    integer variable: i#
$08 <off >off $11

    string variable : s$
$09 <off >off $12

   real matrix      :  r()
$7B indx <off >off $10

   integer matrix   :  i#()
$7C indx <off >off $11

   string matrix    :  s$()
$7D indx <off >off $12

   real literal     :  3.14
$01 (5 byte floating point) $0D

   integer literal :  3
$CE (1 byte) $0D

   integer literal :  278
$02 (2 bytes) $0D

   string literal  :  "hi"
$03 length string $0F

   string literal  :  ""0""
$CD (1 byte) $0F
```

In the above, **indx** is the number of indices in the matrix, <off is the lo byte of the offset in the name table, >off is the hi byte in the same table.

Note that for the last parameter, the last byte is incremented by 6. For example, the following line:
EXEC a(1,"ab","ab")
is tokenized as:

```
00 00 13 1A 00 00 CE 01 0D 03 02 41 42 0F
03 02 41 42 15
```

If it is to be executed by EXCUTE, two extra bytes are added to the end ($FF $33), so that the third byte is $15. This also assumes that "a" is the first entry in the name table (offset = 0).

A short (one PROC) package has been included on TODAY Disk #10, illustrating this technique. The package is called "exeq" (as in USE exeq) and the procedure is called "call". This procedure will execute any PROC which has no parameters. The use of call is:

```
USE exeq
.
.
.
call(string$)
```

where string$ is a string of unshifted characters which represents the name of the PROC which is to be executed

More ►

EXEQ Package - continued

indirectly. The example below is an excerpt from a COMAL 2.0 program.

```
PROC hello
 PRINT "hello"
ENDPROC hello

USE exeq
call("hello")
string$:="hello"
call(string$)
hello
```

This will print "hello" three times. Not an interesting program, but it will serve as an example. LINK in the package from disk with this command:

```
link "pkg.exeq"
```

While this article has only been on calling PROCs from machine language, the information in it can be used to execute other COMAL commands or functions. I hope that this 'trick' will be of some help to other programmers, and I would be happy to hear any comments from the readers on this topic.

[Ed. Note: Ian also cautions us not to CALL a proc that calls itself, or is recursive. M/L programmers should also read the next two articles describing the META package. Although we don't have the source code for META, we feel that it may also use EXCUTE to work its magic.]

You may write to Ian at:

P.O. Box 8487
Saskatoon, Sask.
CANADA S7K 6K5 ■

USE Meta to Evaluate Expressions

by Glen Colbert

Someday, if you haven't already, you may find yourself needing to change a function within a program or to use a complex expression for an input. For example, a program wants you to enter annual income, but you only know the hourly rate. Wouldn't it be nice to enter 11.30*40*52 instead of exiting the program or tearing the house apart looking for a calculator?

The main procedure in the META package is an expression evaluator. With meta, the above problem would be solved like this:

```
USE meta
INPUT "Annual Salary: ":sal$
eval("salary:="+sal$)
PRINT salary
```

Here, the input **SAL$** is tagged onto a variable name **SALARY** and the entire string is executed as if it were a line within the program. The **EVAL** procedure uses the algebraic (TI) hierarchy of operations, simplifying input requirements.

The program META DEMO 1 plots a function on the screen. However, by using the **EVAL** procedure, the function which is plotted is entered as a string. This permits the user to change the function on the fly without having to stop and enter a new function into the program. Quick, easy, and painless. Try it, you'll like it! ■

How to Use the META Package

by David Stidolph

Borge Christensen, founder of COMAL, visited the United States last July. One of the many wonders he brought with him was a small machine code package for COMAL 2.0 that would evaluate commands contained within a string as a valid COMAL command. This package is on TODAY Disk #10.

META allows a program to issue a command that would normally not be allowed within a running program. For example, the following procedure, LIST'IT, will LIST any procedure to disk with the specified filename.

```
PROC list'it
  USE meta // make sure the package
  PAGE //     is linked.
  PRINT AT 8,2: "procedure name:";
  INPUT "": procname$
  PRINT AT 10,2: "name of file:  ";
  INPUT "": filename$
  DELETE filename$
  eval("LIST "+procname$+" """+filename$)
ENDPROC list'it
```

To get started, insert TODAY Disk #10 in your drive and type:

```
LINK "pkg.meta"
USE meta
```

META is a **ROMMED** package. That means it will <u>not</u> be SAVEd along with the program it is LINKed to. If you plan to use META in a program, use the **LINK'IN'META** proc found in **meta demo 1**.

The **EVAL** procedure inside the **META** package will execute any command passed to it as a string. The previous example has a LIST command as the string passed to **EVAL**.

Another use would be to allow a user to type in any valid equation, and have **EVAL** evaluate it over a range of numbers. The following program shows this:

```
USE meta
PAGE
DIM expr$ OF 40
INPUT "f(x)=": expr$
FOR x:=-3 TO 3 DO
  PRINT x,TAB(6),f(x)
ENDFOR x
FUNC f(x)
  eval("y="+expr$)
  RETURN y
ENDFUNC f
```

As you can see the **META** package is quite powerful. You can use it to evaluate complex expressions, issue commands which are normally impossible from within a running program, or even modify the program. Yes, I said modify itself. You can DELete an entire procedure or function with this proc:

```
PROC get'rid(proc'name$)
  USE meta
  eval("DEL "+proc'name$)
ENDPROC get'rid
```

The following line could be used:

```
get'rid("intro") // delete proc intro
```

The only restriction on this package is that you <u>cannot</u> add new variable names to the name table (only modify existing ones), or increase the program size. If you try either of these, the computer will lock up, and destroy your program.

If a normally TRAPpable error occurs during the execution of **EVAL**, the proper error is generated and can be TRAPped.

There is another command within the META package, called **AGAIN**. It executes the last string executed with **EVAL**. ■

Epson Screendump Package

by Dennis Kurnot

This package dumps a graphic screen to an EPSON printer using the CONNECTION printer interface. COMAL's built-in PRINTSCREEN procedure is a wonderful screen dump, and will dump a graphic screen to an EPSON printer using the CONNECTION interface in the emulation mode.

However, the printhead rocks back and forth because of a printer peculiarity when graphics and characters are mixed on the same line. This new package solves the problem. Significant aspects of this new package are as follows:

It is fast because it is written in machine language.

The CONNECTION interface can be left in its emulation mode which is the mode most people seem to use it for general purpose work. This means no switches have to be changed nor do any preliminary software commands have to be sent to the interface before using the SCREENDUMP procedure.

The printer performs a smooth left-to-right pass when printing graphic lines. This is in contrast to routines that mix graphics mode and character mode on the same line causing the EPSON printhead to rock back and forth. This has to be hard on the printer and increases print time.

Blank graphic lines are printed by simply issuing a carriage return & linefeed instead of causing the printhead to make an unnecessary pass across the page. This speeds up graphic dumps that contain blank lines.

The printhead moves across each line only as far as necessary to print the picture. This speeds up the graphic dump.

The package is loaded from disk:

```
link "pkg.eps'grph9000"
```

The package is activated with:

```
use printer
```

The procedure is called with:

```
screendump (density, size)
```

The "density" argument is a number from 0 to 6 that corresponds to the graphic densities described in the EPSON manual. These densities in dots per inch are:

```
0= 60, 1=120, 2=120,
3=240, 4= 80, 5= 72, 6= 90
```

Use density number 5 for true aspect ratio. Check the Epson printer manual for more information including some limitations for double and quadruple densities.

The "size" argument is a 0 or 1. A value of 1 causes the picture to be printed double size. This means one screen dot will be printed as two dots horizontally and two dots vertically. A value of 0 causes the dump to be one printer dot for each screen dot.

Arguments are error checked. If arguments are in error, an error code of one is reported.

The printout is centered on an 8.5 inch wide paper. Some combinations of density and double size cannot be printed on the 8.5 inch wide paper. The package will override such a request and print normal size at the left edge of the paper. The source can easily be modified to be

More ►

centered for all cases on 14 inch wide paper on wide carriage printers.

Jesse Knight's book COMAL 2.0 Packages was used to determine the structure of the machine language program. The mechanism the COMAL developers used to allow the mixing of COMAL and machine language is igenious, versatile, and easy to use. Jesse's book illustrates this mechanism the package could not have been written without it. The Cartridge Tutorial Binder chapter 8 also cqntains information on packages.

The routine CFNAME described in Jesse's book must have the CNAM argument located in lower memory. Location $0334 was used as the starting address to store CNAM before calling CFNAME.

The package sends all necessary control codes. These can easily be changed in the source code. The control codes to the interface and printer can be explained in terms of COMAL statements as follows:

```
// find unused file #, call it X.

OPEN FILE X,"u4:/s0" // talk to interface

PRINT FILE X: chr$(27),chr$(87), //turn
              off 80 column width limit-
              ation in effect on powerup

PRINT FILE X: chr$(27),chr$(76), //
              toggle interface linefeed

CLOSE FILE X

OPEN FILE X, "u4:/s6" // talk to printer

PRINT FILE X:chr$(27),chr$(65),chr$(8), //
set linefeed to 8 dots

PRINT FILE X:chr$(27),"!@", // set
                  printer to pica for a
                  consistant left margin

//calculate left margin to center
  graphic - 'left'

PRINT FILE X:chr$(27),chr$(108),chr$(left)
// set left margin

//print graphic screen at $E000

PRINT FILE X:chr$(27),chr$(65),chr$(12),
// set linefeed to 12 dots

PRINT FILE X:chr$(27),"!A" // leave
                  printer in desired
                  mode (elite here)

CLOSE FILE X

OPEN FILE X,"u4:/s0" // talk to interface

PRINT FILE X:chr$(27),chr$(76), //toggle
                       interface linefeed

CLOSE FILE X

RETURN
```

[Ed.Note- When Dennis called to tell us that his package was done, we asked him to assemble the code in an area other than the normal $8009 so that it could be used with other packages. He did so, and you will find two versions of his package on TODAY Disk #10. One is located at $9000, as described above and also one located at $8009.

We urge package writers to follow his lead. $8009 is getting crowded. If you assemble your code to locate itself other than $8009, please include the address as part of the filename.] ■

Bitmap Compression - A New COMAL Package

by David Stidolph

In Volume 6, Issue 4 of the Transactor, Chris Zamara published a machine code program to shrink BASIC bitmap pictures by coding them. I took his idea and wrote both a COMAL 2.0 package and COMAL 0.14 machine code routine that does this. Demonstration programs, sample pictures, and the machine code are on TODAY Disk #10.

The idea of compression is simple. Much of any picture is blank space (all bytes on). To shrink the bitmap, the bytes are encoded. When three consecutive bytes or more are the same, we code them. First we send a byte 254 followed by the byte to repeat, and finally the number of bytes to show. For instance, 100 consecutive bytes of zero's would be coded 254, 0, 100. This would save 97 bytes. If a single byte 254 is encountered, we code 254, 254, 1.

Random pictures will not shrink much, but most files are reduced by 40-80%. COMAL hires pictures may also be compacted.

The package's filename is pkg.compactor. To use this package, first do this:

```
link "pkg.compactor"
use compactor
```

Two PROCs do the work. Before you call the procs, OPEN a file to the drive, then pass the open file number as a parameter.

```
save'compact(file'num)
load'compact(file'num)
```

Programming examples may be found in the programs described on the next page. Files made with this system will have .crg as a suffix to their filename. ■

```
; compile/uncompile pictures
; by david stidolph
;
        .lib c64symb
        opt list
        *= $be00
        .byt defpag
        .wor end
        .wor dummy
;
        .byt 9,'compactor'
        .wor procnm
        .wor dummy
        .byt 0
;
procnm .byt 12,'save'
        .byt 39,'compact'
        .wor savehd
        .byt 12,'load'
        .byt 39,'compact'
        .wor loadhd
        .byt 0
;
savehd .byt proc
        .wor savecd
        .byt 1
        .byt value+int
        .byt endprc
;
loadhd .byt proc
        .wor loadcd
        .byt 1
        .byt value+int
        .byt endprc
;
; picture compressor/decompressor
; idea for project from:
;   article by chris zamara
;   transactor -- volume 6
;   issue 4    -- jan 1986
;
; code for comal, written by
;               david stidolph
;               nov 12,1985
;
byte   .byt 0
oldbyt .byt 0
last   .byt 0
num    .byt 0
;
loadcd lda #1
        jsr fndpar
        ldy #1
        lda (copy1),y
        tax
        jsr cchkin
        lda #$00
        sta copy1
        lda #$e0
        sta copy1+1
;
mainld jsr crdt
        cmp #254
        bne normal
        jsr crdt
        sta byte
        jsr crdt
        sta num
        ldy #00
next    lda byte
        sta (copy1),y
        jsr incrcv
decnum dec num
        bne next
        jmp mainld
;
normal ldy #00
        sta (copy1),y
        jsr incrcv
        jmp mainld
;
incrcv inc copy1
        bne notr1
        inc copy1+1
notr1  lda copy1+1
        cmp #255
        bne ntendr
        lda copy1
        cmp #64
        bne ntendr
        pla
        pla
        jmp cclrch
ntendr rts
```

```
; compactor for comal 2.0
;
savecd lda #1
        jsr fndpar
        ldy #1
        sty num
        lda (copy1),y
        tax
        jsr cckout
;
        lda #$00
        sta copy1
        lda #$e0
        sta copy1+1
;
        sei
        lda 1
        and #%11111101
        sta 1
        ldy #0
        lda (copy1),y
        sta byte
        lda 1
        ora #%00000010
        sta 1
        cli
        inc copy1
;
savemn ldy #00
        lda byte
        sta oldbyt
        sei
        lda 1
        and #%11111101
        sta 1
        lda (copy1),y
        sta byte
        lda 1
        ora #%00000010
        sta 1
        cli
        jsr putout
        inc copy1
        bne notnow
        inc copy1+1
notnow lda copy1+1
        cmp #$ff
        bne savemn
        lda copy1
        cmp #64
        bne savemn
;
; end compress
;
        jsr nmatch
finish jmp cclrch
;
putout lda byte
        cmp oldbyt
        bne nmatch
        inc num
        bne retrn
        dec num
        jmp not2
retrn  rts
;
nmatch lda num
        cmp #1
        bne not1
        jsr send
        rts
;
not1    cmp #2
        bne not2
        jsr send
        jsr send
        lda #1
        sta num
        rts
;
not2    lda #254
        jsr cwrt
        lda oldbyt
        jsr cwrt
        lda num
        jsr cwrt
        lda #1
        sta num
        rts
;
send    lda oldbyt
        cmp #254
        bne not254
        jsr cwrt
        lda #254
        jsr cwrt
        lda #1
not254 jsr cwrt
        rts
;
end     .end
```

CRG - Bitmap Compression SYSTEM

by Colin Thompson

The compression routines explained on the opposite page have been used to form the heart of several application programs you will find on TODAY Disk #10.

The programs are written in both COMAL 2.0 and COMAL 0.14. Both systems allow you to convert COMAL bitmaps to compacted bitmaps, view the compacted bitmaps, and update the two SEQ files needed by the programs.

The 2.0 system is a single program, called **CRG2**. The package **pkg.compactor** is LINKed to the COMAL program, and is also recorded on the disk. When you load and run CRG2, you will be presented with the following menu options:

C compact a bitmap
V view a compact pix disk
U update the " compact pix" file
D update the " directory" file
0 directory of drive 0
1 directory of drive 1

If you have a dual drive, the last option will appear. This system depends on three things to work properly:

1. A supply of COMAL bitmap pictures, properly named with the .hrg filename suffix. A few bitmaps are included on TODAY Disk #10.

2. A SEQ file called " directory" that holds the filenames of the bitmap pictures on the disk. The filenames must have the suffix .hrg stripped off.

3. A SEQ file called " compact pix" that holds the filenames of the compacted pictures on the disk. The filenames must have the suffix .crg stripped off.

The necessary files are included on TODAY Disk #10. To use this system, first copy the needed files to another disk.

Operating instructions. Select option v to see the compacted pictures on the disk. Use the cursor keys to select the picture to view and press return to load it. After the picture is loaded, press f1 to return to the menu, or press f5 to see the picture. Press r to reverse the picture. Press p to print the picture to your 1525 type printer. Press s to save a copy of the picture as a COMAL 2.0 savescreen. To Exit, press the STOP key.

Select option c to convert pictures to compacted form. You can compact a COMAL HIRES picture, or standard COMAL bitmaps. If you have a single drive, you will be prompted to insert the **source disk** and then after the source picture is loaded, to insert a formatted **target disk**. Dual drive users must put the **source disk** in drive 0 and the **target disk** in drive 1.

After the conversions are complete, select option u and follow the instructions. Option d is used after you have assembled some bitmaps on a disk for conversion.

The COMAL 0.14 version is similar in operation, but is split up into 3 programs to handle the work. **CRG14.COMPACTOR** will convert bitmaps. **CRG14.VIEWER** will show you the pictures. **CRG14.FILEWRITER** will produce both SEQ files needed.

Two disks full of bitmaps, called SLIDE SHOW #1 & #2, are available for $9.75 each to COMAL Today subscribers. Each disk has 18 bitmaps, and viewing software.

COMAL 2.0 programmers please see the two functions on page 61. The functions, when used with pkg.compactor, will load or save compacted pictures, and if an error is detected, will properly report it.■

Comparing Disk Files

by Len Lindsay

Did you ever make a copy of a file, and then wish you could verify that the copy was really identical? Did you ever have two files on two different disks, and wonder if they were the same? We have those same kind of problems here. So I wrote a short COMAL 2.0 program that will copy a file, and then verify it. It works with single drives as well as dual drives, but the files must be less than 28K long (all the memory left free).

This program not only compares two files, but prints the characters (and their ASCII values) in both files side by side - pointing out any mismatches! Non printing characters are printed as a period.

You can use this program to make a duplicate copy of a file on the same disk as the original file. Just use a different name for the target file.

If you use two disks with a single drive, you will have to swap disks (the program tells you when to swap the disks). If both files are on the same disk, just keep that disk in the drive the whole time - hit RETURN at each prompt to insert a disk.

IMPORTANT NOTE: To copy a PRG type file, you must include a ",PRG" as the last part of the target filename: **MENU.BACKUP,PRG**

POSSIBLE VARIATIONS

Printing the contents of the files on the screen takes time. If you don't wish to see the contents displayed (and thus speed up the process), just add the following line as the first line in the procedure named PRINTOUT:

```
RETURN
```

If you want to copy files but not compare them, just delete the line that calls the COMPARE procedure as well as deleting the whole COMPARE procedure itself:

```
DEL COMPARE
DEL 50 //as in the listing numbered by 10
```

If you want to just compare files (no copying), delete the COPY'FILE procedure and the line that calls it:

```
DEL COPY'FILE
DEL 40 //as in the listing numbered by 10
```

The program does not check if the files exist before comparing them. You may wish to add the **FILE'EXISTS** function (listed in COMAL TODAY #6 page 42) or use the **TRAP** structure. But remember, the more you add to the program, the less memory will be free for copy and compare storage.

```
//delete"0:copy/compare" // comal 2.0 only
//save"0:copy/compare"
init
copy'file(source$,target$,max'size)
compare(source$,target$,max'size)
//
PROC init
  PAGE
  max'size:=28*1024
  DIM source$ OF 23, target$ OF 23
  INPUT "Source Name: ": source$
  INPUT "Target Name: ": target$
ENDPROC init
//
PROC copy'file(source$,target$,max) CLOSED
  DIM image$ OF max, dummy$ OF 0
  read'it'in
  IF LEN(image$)>=max THEN END "Too long"
  INPUT "Insert target disk:": dummy$
  write'it'back
  //
```

More ►

```
  PROC read'it'in
    OPEN FILE 5,source$,READ
    WHILE NOT EOF(5) DO
      image$:+GET$(5,254)
    ENDWHILE
    CLOSE
  ENDPROC read'it'in
  //
  PROC write'it'back
    MOUNT
    length:=LEN(image$)
    OPEN FILE 5,target$,WRITE
    count:=1
    WHILE count+253<length DO
     PRINT FILE 5:image$(count:count+253),
     count:+254
    ENDWHILE
    PRINT FILE 5: image$(count:length),
    CLOSE
  ENDPROC write'it'back
ENDPROC copy'file
//
PROC compare(name1$,name2$,max) CLOSED
  DIM image$ OF max,text$ OF 1,dummy$ OF 0
  OPEN FILE 2,name2$,READ
  WHILE NOT EOF(2) DO
    image$:+GET$(2,254)
  ENDWHILE
  CLOSE FILE 2
  INPUT "insert source disk:": dummy$
  OPEN FILE 3,name1$,READ
  errors:=FALSE; byte:=0
  WHILE NOT (EOF(3) OR byte=LEN(image$)) DO
    byte:+1
    text$:=GET$(3,1)
    IF text$<>image$(byte) THEN errors:+1
    printout
  ENDWHILE
  PRINT "Errors found:";errors
  IF NOT EOF(3) THEN
    PRINT "File";name1$;"is longer."
  ELIF byte<LEN(image$) THEN
    PRINT "File";name2$;"is longer."
  ELIF NOT errors THEN
    PRINT "The files matched."
  ENDIF
  CLOSE
  //
  PROC printout
    //RETURN//add this line to skip print
    PRINT USING "#####>": byte;
    print'char(ORD(image$(byte)))
    print'char(ORD(text$))
    IF errors THEN PRINT "Mismatch";
    PRINT // carriage return
  ENDPROC printout
  //
  PROC print'char(n)
    IF (n>=32 AND n<128) OR n>=160 THEN
      PRINT CHR$(n);
      PRINT USING "###": n;
    ELSE
      PRINT USING ". ###": n;
    ENDIF
  ENDPROC print'char
ENDPROC compare
```

Unit to Unit File Copier

This small program can copy programs using two 1541 drives. One of the drives must be set to device 9. It displays the directory of the disk in the unit 8 drive. Then it asks for the filename. To copy a PRG type file include ",PRG" after the file name. The file is then written to drive 2 (unit 9 drive 0 is referred to as drive "2:"). Unit 9's disk directory is then displayed. Easy! This is the whole program:

```
// copy files unit to unit - max 14K
DIR "0:*"
INPUT "File name: ": name$
OPEN FILE 2,"0:"+name$,READ
OPEN FILE 3,"2:"+name$,WRITE
PRINT FILE 3: GET$(2,14000),
CLOSE
DIR "2:*"
```

Also see a related program on page 42.

New Screendump Package Handles Three Printers Okidata 93a, Epson RX80, Gemini 10x

by Randy Finch

The FINCHUTIL package on TODAY DISK #10 includes a hi-res screen dump for Epson RX-80, Star Gemini-10X, and Okidata 93A or compatible printers. There are four different screen dumps for each printer.

HOW TO USE

To use the package in your program you must first load your program. Next, insert TODAY DISK #10 in your disk drive and type the following:

```
link "pkg.finchutil"
```

This will attach the package to your program. If you save your program at this point, the package will be saved along with it. The following statement must be in your program before using any of the procedures in the package:

```
use finchutilities
```

This will make the package known to the program.

PROCEDURES

There are three procedures in this package:

```
gl0xdump(int)
rx80dump(int)
okidump(int)
```

The first two execute the same code because the printers are graphics compatible. I included two procedure names because I personally have a Gemini-10X, but I thought there would be many Epson users who would prefer using rx80dump instead of gl0xdump.

Each of the three procedures requires one integer parameter. If you pass a floating point number, COMAL converts it for you. This integer must be either a 1,2,4, or 8; each representing a different type of hi-res screen dump. Examples:

```
gl0xdump(1)
rx80dump(4)

a#=2
okidump(a#)
```

TYPES OF SCREENDUMPS - Epson/Star

The following dumps are sent to the the Gemini-10X, Epson RX-80 or compatible printer according to the integer passed as a parameter.

Parameter = 1

Sends a small dump (1/4 page) to the printer in single density mode.

Parameter = 2

Sends a small dump to the printer in double density mode.

Parameter = 4

Sends a large dump (full page) to the printer in single density mode.

Parameter = 8

Sends a large dump to the printer in double density mode.

TYPES OF SCREEN DUMPS - Okidata

The following dumps are sent to the Okidata 93A or compatible printer according to the integer passed as a parameter.

More ►

Parameter = 1

Sends a small dump (1/4 page) to the printer in the 60 dots-per-inch (dpi) mode.

Parameter = 2

Sends a small dump to the printer in the 72 dpi mode.

Parameter = 4

Sends a big dump (full page) to the printer in the 60 dpi mode.

Parameter = 8

Sends a big dump to the printer in the 72 dpi mode.

The package uses the transparent mode on the Cardco Card?/A interface which is a secondary address of 5. If your secondary address is different, type the following in direct mode before you save your program with the package attached:

```
sec'addr=3 // use appropriate number
poke $83c5, $60 + sec'addr
poke $8424, $60 + sec'addr
```

This will ensure that the proper secondary address is used for your interface. ■

UPCOMING on TODAY Disk #11:

The GRAPHICS Editor. A complete COMAL 0.14 bitmap editing system, compatible with the CRG14 programs found on TODAY Disk #10.

Available Drives

by David Stidolph

This short program shows how to use the procedure GET'DRIVES. This procedure will show what drive units are available. Remember that 0: and 1: refer to disk unit number 8, drives zero and one.

This procedure uses an 8 element array numbered 0 to 7 (drive number go from 0 to 7). It will set each position to TRUE if the drive exists, otherwise set it to false.

```
// delete "show'drives" // COMAL 2.0 only
// save   "show'drives"
//  by David Stidolph
//
PAGE
DIM valid'drive(0:7)
PRINT "Testing drives, please wait..."
get'drives(valid'drive())
PRINT
PRINT "Valid drive units are:"
PRINT "----------------------"
FOR x:=0 TO 7 DO
  IF valid'drive(x) THEN
    PRINT "UNIT """,x,":"""
  ENDIF
ENDFOR x
END
//
PROC get'drives(REF drive()) CLOSED
  FOR x:=0 TO 7 DO
    TRAP
      MOUNT STR$(x)+":"
      drive(x):=TRUE
    HANDLER
      IF ERR=274 OR ERR=208 THEN
        drive(x):=FALSE
      ELSE
        drive(x):=TRUE
      ENDIF
    ENDTRAP
  ENDFOR x
ENDPROC get'drives
```

■

M/L Monitor Package for COMAL 2.0

by Glen Colbert

Jesse Knight included a machine code monitor with his book/disk COMAL 2.0 PACKAGES. I have re-written the monitor so that it works more closely with COMAL 2.0, and provides some new features. The monitor is called pkg.cmon and is on the 2.0 side of TODAY DISK #10.

To load the monitor into memory you issue the following command with TODAY DISK #10 in the disk drive:

```
link "pkg.cmon"
```

Once loaded you have to add the command words to the name table with the USE command.

```
use cmon
```

Now the commands within the package CMON are available for use (including within a program -- which also has to execute the USE cmon command to use them). The command list follows.

MONITOR - This command takes you out of the COMAL editor and puts you in a machine code monitor. The commands of this monitor will be listed later in this article.

LOCATE'REAL(num) - This function returns the actual memory address of a REAL number variable.

LOCATE'INT(num#) - This function returns the actual memory address of an INTEGER variable.

LOCATE'STR(string$) - This function returns the actual memory address of a STRING variable.

HEX$(num) - This function returns a string containing the HEXIDECIMAL equivalent of a number. The string is always 5 characters long, with a dollar sign ($) first followed by four HEX characters (0-f). You can use SUBSTRING notation to get only the characters you care about.

BIN$(num) - This function returns a string containing the BINARY equivalent of a number. The string is always 18 characters long, with a space and percent sign in front (%) followed by 16 BINARY characters (0 or 1). You can use SUBSTRING notation to get only the characters you need.

THE MONITOR

The machine code monitor PKG.CMON resides at $a800 and uses the RS-232 buffers. Do not use the USER port while running this monitor.

Here is a list of the MONITOR commands:

A - Assemble code line
    (.a 8009 lda #$01)
D - Disassemble code line
    (.d 8009)
F - Fill range of memory
    (.f start,end,value)
H - Hunt for values in range
    (.h start,end,value)
L - Load PRG file from disk
    (.l "0:filename",08)
M - Dump memory range
    (.m 8000,8100)
R - Show all registers and cart page#
    (.r)
S - Save memory to disk
    (.s "0:filename",08,start,new)
T - Transfer range to new location
    (.t start,end,new): .t 0800,0810,0900
X - Exit monitor and return to COMAL
    (.x) ■

Now Available Through Aquarian Software

Gold Disk Series

Volumes 1 through 11 Now Available!!!

Volume 11 Features a C-64 Assembler

Each Disk Contains:

- The Monthly Feature Program
- Programming Tutorials
- High Quality Games
- And Much More

Gold Disk Series for 128
Coming Soon!

Only $14.95 Per Disk*

* Plus Shipping and Handling

The Cataloger

The Ultimate Disk Cataloging System for the 64!

Features of The Cataloger V3.5A include:

- ★ Loads directly from the disk itself.
- ★ Ability to change name of entry.
- ★ Fast — Uses relative files exclusively
- ★ Search, Sort and Print by any of 12 fields.
- ★ 1100-program (or disk) capacity per data disk.
- ★ All machine language.
- ★ Menu driven — very easy to use.
- ★ Works with one or two drives.

Only $24.95

BobsTerm Pro

The Ultimate Terminal Software !

Upload / Download Supports Punter, X-Modem, XON / XOFF, DC1 / DC2, and Much More!

28.5 Byte Buffer with unmatched editing abilities

- User Adjustable Parameters
- 10 Custom Character Sets
- Unlimitied Phone Book Storage
- Programmable Macro Command Strings

Only $59.95

MATRIX — NOW AVAILABLE!!

The Indispensable C-128 Utility / Starter Kit !

Use **dozens** of 128 features in the 64 mode:

- Numeric Key Pad
- Cursor Keys
- 80-Column RGB Output
- Many Other Special Function keys

One-Key Functions Include:

- **2 Megahertz "Fast Mode"**
- **One-Key Screen Dumps**
- **Full-Featured DOS Utility Menu**

Other Features Include:

- Fast Loading
- Fast Copy For The 1571!
- Relocatable In Memory
- 100% Transparent to BASIC

Available Now For Only $59.95

Graphic Screen Exporter

A Universal Graphics Converter !

Converts Anything to Anything — Including:

Koala Pad	Doodle
Flexidraw	Print Shop
COMAL	Paint Magic
CAD GEM	Micron Eye

And Many Many More ! !

The Most Versatile Graphics Utility Ever Released for the Commodore 64 !

Only $29.95

ALSO AVAILABLE:

OmiTerm.....$19.95
Full-Feature Terminal at an Affordable Price!

Turbo Calc/64.....$17.95
A great spreadsheet at an Unbelievable Price!

Tax Computation.....$29.95
The friendliest tax package on the market.

Guitar Master.....$49.95
A comprehensive musical instruction package

Fast Boot!.....$14.95
Mike J. Henry's Fast Loader for 1541/MSD

Thriller Collection.....$24.95
Seven intricate text adventures on one disk

Call or Write for Full Catalog !

CAD-GEM

Computer Assisted Design
Graphic Element Manipulation

A Wire Frame CAD system for the C64 !
Input from Joystick, Track Ball, Light Pen or Graphics Tablet
360 Degree Rotation in .1 Degree Increments
Scaling on a 64K x 64K, 2048 Mega-Bit Virtual Screen
Independent Manipulation of 400 Objects (Points or Lines)

You must see CAD GEM to believe it!
Demo Disk Available for $3.00

$89.95

MODEM MASTER

The Friendliest Commodore BBS Available

- Works with 1541 or MSD Dual Drive
- 300 / 1200 Baud Operation
- New Punter File Transfer Protocol
- Sub-Directories for File Transfer
- 250 User Capacity
- Accurate Clock / Calendar
- Printer Output
- Information Files
- "Old" E-Mail Deleted After One Week
- Set Up In Only 10 Minutes !

Only $29.95

Total Software Development System

by Kevin Pickell
Now Available in the States !

Assembler/Editor — fast load, get, log and loadat; adds 38 new commands; full macro instructions; allows 13-character labels; assembles to and from disk
Sprite Editor — 256 sprites in memory, view 64 at same time, works with keyboard, joystick or trackball, animates sprites during design
Unassembler — create source code from any ML program
Sound Editor — create interrupt-driven sound effects
Character Editor — edit all characters. Screens to 255x64. Hi-res & Multi-color Character Sets

TSDS automatically includes sprites, characters, mattes and sound effects into source code!

Only $39.95

128 Version Coming Soon !

Aquarian Software
P.O. Box 22184
Portland, OR 97222

VISA

To order, Call: (503) 654-2641
VISA & MasterCard Accepted

MasterCard

Add 3.00 S & H Per Order
(Add Additional $2.00 for COD)
Canadian Orders Add 10.00 S&H
Allow 3-4 Weeks For Delivery

Write or Call for Full Catalog — Dealer Inquiries Welcome !

VALGOL: for Non-Programmers

From its modest beginnings in Southern California's San Fernando Valley, Valgol is enjoying a dramatic surge of popularity across the industry.

Valgol commands include **really**, **like**, **well**, and **y*know**. Variables are assigned with the **=like** and **=totally** operators. Other operators include the California Booleans, **fersure** and **noway**. Repetitions of code are handled in **for/sure** loops. Here is a sample Valgol program:

```
like y*know (I mean) start
if pizza =like bitchen and
  b =like tubular and
  c =like grodyax
then
  for I =like to oh maybe 100
    do wah -(ditty)
    barf(1) =totally gross(out)
  sure
like bag this problem
really
like totally (y*know)
```

Valgol is characterized by its unfriendly error messages. For example, when the user makes a syntax error, the interpreter displays the message:

gag me with a spoon

The preceding informative announcement of this revolutionary new language for non programmers was first seen in the November 1984 issue of the University of Waterloo's mathNEWS. Transactor popularized it a year later. We now are bringing it to the attention of COMALites around the world to show how COMAL is influencing these new powerful languages. Notice the automatic indentation of structures, long variable names, and descriptive error messages. ■

Your Orders

All orders are usually processed within a week. However, due to some bad checks we have recieved, orders paid by check are held for 2 weeks to allow the check to clear the banks. We do not accept COD orders. Purchase orders are accepted only if payment accompanies the order. We make an exception for schools. We will accept a school order if payment is guaranteed in 30 days. Remember to add in the shipping charges. You may order by phone if you have a Visa or MasterCard to charge it to. On charge orders, the shipping is calculated automatically for you by our order processing system. Our order processing system is a large COMAL program - nearly 3000 lines long. ■

Filename Conventions

Here are the filename conventions:

Suffixed	Prefixed	Meaning
NAME	NAME	COMAL program file
NAME.L	LST.NAME	Program listed to disk
NAME.PROC	PROC.NAME	PROC listed to disk
NAME.FUNC	FUNC.NAME	FUNC listed to disk
NAME.DAT	DAT.NAME	Data file
NAME.TXT	TXT.NAME	Text file
NAME.DOC	DOC.NAME	Documentation file
	EXT.NAME	External PROC/FUNC
	SHAP.NAME	Sprite shape file
	FONT.NAME	COMAL font file
	FONT.MC.NAME	Multicolor font file
	SET.NAME	Basic type font file
	PKG.NAME	Package file
	BAT.NAME	Batch file
	SNG.NAME	Song file
	HRG.NAME	Color COMAL picture
NAME.HRG		Black/White bitmap
	CRG.NAME	Compacted color pix
NAME.CRG		Compacted B/W bitmap

Disk Directory

```
Disk: user group 11

>--------------<
!   programs   !
>--------------<
bitmapdoodle&2.0
color'hanoi
plot'a'function
roman'numerals
the'memory'game
>--------------<
! music and    !
!  graphics   !
>--------------<
dizzy'turtle
drawto
kaleidoscope
paint'by'letter
raster'scanner
sampler
>--------------<
!  data files  !
!--------------!
! do not load  !
>--------------<
dat.bwv779
dat.bwv781
dat.bwv783
dat.bwv786
dat.bwv794
dat.bwv801
mem.raster

>--------------<
! 1520 plotter !
>--------------<
1520draw'house
kaleidoscope1520
>--------------<
!external proc !
!   examples   !
>--------------<
main'program
dsp.circles
dsp.hobie
dsp.hobie2
dsp.jane
dsp.my'moire
dsp.turtlejoy
dsp.turtlejoy2
dsp.wild
ext.circles
ext.hobie
ext.hobie2
ext.jane
ext.my'moire
ext.turtlejoy
ext.turtlejoy2
ext.wild
>--------------<
!2.0 accounting!
!    system    !
!--------------!
!copy the files!
!below to a new!
!    disk      !
>--------------<

>--------------<
!master program!
>--------------<
main'menu
>--------------<
! sub-programs !
>--------------<
create.categorys
do'posting
expense'record
income'record
init'expense
init'income
init'post
>--------------<
!  data files  !
>--------------<
cardv85
categories
expnames
hg85
inc85
incnames
jb85
post85
year'names
>--------------<
!  the above   !
! files should !
!be transferred!
!   together   !
>--------------<
1 blocks free.
```

Easy Curves

1) Insert your COMAL disk in drive*.
2) Type LOAD "C64 COMAL*",8
3) Type RUN (starts COMAL)
4) Type AUTO (turn on auto line#'s)
5) Enter the program lines shown below (COMAL indents lines for you)
6) Hit RETURN key twice when done
7) Type RUN

```
0010 setup
0020 curve
0030 paint'it
0040 add'words
0050 //
0060 proc setup
0070   black:=0; yellow:=7
0080   background black
0090   pencolor yellow
0100   setgraphic 0 //hi res screen
0110   hideturtle
0120 endproc setup
0130 //
0140 proc curve
0150   moveto 110,0
0160   drawto 110,199
0170   for row:=0 to 10 step .03 do
0180     drawto 110+99*sin(row),row*20
0190   endfor row
0200 endproc curve
0210 //
0220 proc paint'it
0230   fill 120,20
0240   fill 100,90
0250   fill 120,180
0260   fill 100,198
0270 endproc paint'it
0280 //
0290 proc add'words
0300   pencolor black
0310   background yellow
0320   plottext 120,155,"comal is a"
0330   plottext 16,90,"programmers"
0340   plottext 120,30,"paradise"
0350 endproc add'words
```

Notice how easy graphics are in COMAL. Lines 70-100 set up the screen colors. Lines 150-190 draw on the screen. Lines 230-260 fill (paint) whole parts. Even putting text on the graphic screen is easy. See lines 320-340. All this is standard and built in as part of COMAL. Plus a full turtle graphics system. Now you know why there are 100,000 users.

Price Protection

Every COMAL TODAY subscriber is protected. If you buy any COMAL item from us, and the price goes down within 1 month, you automatically are entitled to a credit for the difference. Now you can buy with confidence. We wonder how many other computer companies will be willing to follow our lead with this policy. Note: special prices offered at a show, conference, or similar event do not count as a price drop.

DISCOUNTS ON NEARLY EVERYTHING

All COMAL TODAY subscribers now automatically get a discount on nearly everything they buy from us. We will continue our special quantity prices to schools and groups as well.

THE NEW PRICE STRUCTURE

Keep it simple. What good is a discount if it takes you an hour to calculate? Effective December 1, 1985, this is how our SUBSCRIBER DISCOUNT works:

$1 off each CHEATSHEET keyboard overlay
$1 off each back issue of COMAL TODAY
$2 off every book (now includes all books)
   only $4.95 for optional matching disk
$4 off COMALite shirts
$4 off each COMAL 2.0 Deluxe Cartridge Pak
All other disks are $9.75 each
$10 off each McPen Lightpen
$15 off the set of 4 Cartridge Demo Disks

CREDITS AVAILABLE - PROTECTION PLAN

To claim your credit you must return your original invoice showing your purchase at the higher price. Circle the items and clearly mark the credit amount. The credit can be applied to any future order. ■

Disk Directory

Disk:font.master #1

```
boot
font editor
 comal fonts
>-------------<
! comal fonts  !
>-------------<
font.80column
font.art deco.pb
font.ascii
font.colin
font.computer
font.d&d1
font.fancy
font.french
font.gothic
font.greek
font.greek lang
font.hebrew
font.italic.asci
font.music notes
font.outline
font.repton
font.roman
font.roman fancy
font.round
font.russian
font.san quentin
font.script
font.streamline
font.tech1
font.tech2
font.thick chars
font.thin europe
font.type
font.typeset
font.underline
font.warbot
>-------------<
!  multicolor  !
!     fonts    !
>-------------<
font.mc.shaded
font.mc.shaded1
font.mc.square
>-------------<
! font disk #1 !
!copyright 1985!
! comal users  !
!  group,usa   !
!-------------!
!  may not be  !
!  copied or   !
!  placed in   !
!  user group  !
!  libraries   !
>-------------<
10  blocks free.
```

Disk: font disk #1 side 2

```
boot
font viewer
singlecopy
information
type old english
edit unprotected
>-------------<
! comal fonts  !
>-------------<
font.mirror
font.old english
font.pattern
font.pet/graphic
font.standard
font.thin char
font.vic20
>-------------<
!comal 2.0 demo!
!   program    !
>-------------<
rotate.pkg.demo
>-------------<
! do not load  !
>-------------<
pkg.rotate
 basic fonts
 comal fonts
>-------------<
! basic fonts  !
!-------------!
!may be used by!
! comal 0.14   !
!program below !
>-------------<
set.80column.b
set.art deco.a
set.art deco.b
set.colin.a
set.colin.b
set.computer.a
set.computer.b
set.d&d1.a
set.fancy.b
set.french.b
set.gothic.b
set.ital.nocur.b
set.italic.b
set.music note.b
set.outline.b
set.repton.a
set.rom fancy.b
set.roman.a
set.round.a
set.round.b
set.sanquentin.a
set.script.b
set.streamline.a
set.tech1.a
set.tech2.a
set.thick chrs.a
set.thin europ.a
set.thin europ.b
set.type.b
set.typeset.a
set.typeset.b
set.underline.b
set.vic20.a
set.vic20.b
set.warbot.a
>-------------<
!  comal 0.14  !
! font viewer  !
>-------------<
view'font/demo
5  blocks free.
```

Order Form

Name:_____ SUBSCRIBER NUMBER:_____ Dec. 1985
(required for reduced prices-except new subs)
Street:_____ Pay by check/MoneyOrder in US Dollars
Canada Postal US Dollar Money Order is OK
City/St/Zip:_____ VISA / MasterCard print card#/exp date:

VISA/MC #:_____exp date:_____Signature:_____

Qnty Price List/Subscriber price-Item Description (all disks Commodore 1541 format) Prices subject to change

SYSTEMS: (two disks may be supplied as a double sided disk)

[ ] _____ $98.95/$94.95 Deluxe Cartridge Pak (2 books/1 disk/1 cartridge)-(shipping add $4)
[ ] _____ $14.95/$12.95 COMAL Quick 0.14 (fastloaded) with Utility Disk #2 & book (shipping add $2)
[ ] _____ $11.00/$11.00 C64 COMAL 0.14 $11 Special AutoRun/Tutorial disk, COMAL From A To Z
[ ] _____ $350/$350 IBM Denmark PC COMAL (Danish manual/COMAL Handbook)-(shipping add $5)

SUBSCRIPTIONS:

[ ] _____ COMAL TODAY newsletter-> How many issues? ____ Start with: 6 7 8 9 10 11<-Circle one
($14.95 first 6; $2 each added issue; >>> Canada add $1 per issue; >>> overseas add $5 per issue)
[ ] _____ $14.95/$12.95 First 4 issues COMAL TODAY spiral bound (shipping add $2)
[ ] _____ $3.95/$2.95 COMAL Today backissue: circle issues wanted-> 5 6 7 8 9-(shipping add $1 each)
[ ] _____ TODAY DISK subscription >>>>>>>>> How many disks? _____ Start with disk#_____
($49.95/$29.95 for first 6 disks - $5 each added disk)-(no extra shipping charge)

BOOKS: (optional matching disks are $14.95/$4.95)-(>>>>Canada shipping add $1 more per book<<<<)

[ ] _____ $18.95/$16.95 COMAL Handbook - optional disk [___] (shipping add $3)
[ ] _____ $15.95/$13.95 Starting With COMAL (matching disk not available yet)-(shipping add $3)
[ ] _____ $19.95/$17.95 Foundations With COMAL - optional disk [___] (shipping add $3)
[ ] _____ $28.95/$26.95 Structured Programming With COMAL - optional disk [___] (shipping add $3)
[ ] _____ $20.95/$18.95 Beginning COMAL - optional disk [___] (shipping add $3)
[ ] _____ $17.95/$15.95 C64 Graphics With COMAL 0.14 (disk not available yet)-(shipping add $3)
[ ] _____ $6.95/$4.95 COMAL From A To Z (part of $11 special)-(shipping add $2)
[ ] _____ $14.95/$12.95 Captain COMAL Gets Organized (includes disk)-(shipping add $2)
[ ] _____ $6.95/$4.95 COMAL Workbook (perfect companion to Tutorial Disk)-(shipping add $2)
[ ] _____ $14.95/$12.95 COMAL Library of Functions & Procedures (includes disk)-(shipping add $2)
[ ] _____ $14.95/$12.95 Graphics Primer (includes disk) (for COMAL 0.14)-(shipping add $2)
[ ] _____ $6.95/$4.95 Cartridge Graphics & Sound (part of Deluxe Cartridge Pak)-(shipping add $2)
[ ] _____ $19.95/$17.95 COMAL 2.0 Packages (disk includes C64SYMB & Monitor)-(shipping add $2)
[ ] _____ $19.95/$17.95 Packages Library (with disk)(late Jan 86 release)(shipping add $2)
[ ] _____ $19.95/$17.95 Cartridge Tutorial Binder (with disk)(part of Deluxe Cart Pak)(shipping add $3)

DISKS:

[ ] _____ $14.95/$9.75 Font Disk for COMAL 0.14 & 2.0 (40 different fonts plus support programs)
[ ] _____ $94.05/$89.95 19 Disk Set (about 1000 programs for COMAL 0.14)-(shipping add $3)
[ ] _____ $14.95/$9.75 Best of COMAL 0.14 (includes Disk Data Base)
[ ] _____ $7/$7 Auto RUN Demo and Tutorial Disk (part of $11 Special)(perfect with COMAL Workbook)
[ ] _____ $14.95/$9.75 Bricks Tutorials (2 sided BEGINNERS disk)
[ ] _____ $14.95/$9.75 Utility Disk #1 for COMAL 0.14
[ ] _____ $14.95/$9.75 TODAY Disk-Circle disks wanted: 1 2 3 4 5 6 7 8 9 10 (see subscription above)
[ ] _____ $10/$9.75 User Group Disks-Circle disks wanted: 1 2 3 4 5 6 7 8 9 10 11
[ ] _____ $14.95/$9.75 Slide Show Picture Disks-Circle disks wanted: 1 2
[ ] _____ $14.95/$9.75 Cartridge Demo Disks-Circle disks wanted: 1 2 3 4 (or see next line)
[ ] _____ $29.95/$14.95 Set of all Cartridge Demo Disks (includes #1 #2 #3 and #4)
[ ] _____ $14.95/$9.75 Specialty Disks (0.14 & 2.0): [__] Games Disk [__] Typing Disk [__] Modem Disk
Jan. '86 Jan. '86

OTHER:

[ ] _____ OTHER: _____
[ ] _____ $3.95/$2.95 Keyboard Overlay for C64 COMAL 0.14 - Cheatsheet (shipping add $1)
[ ] _____ $49.95/$39.95 McPen Light Pen (with COMAL 2.0 Demo Disk)-(shipping add $3)
[ ] _____ $9.95/$5.95 Royal Blue COMALite Shirt-circle ADULT size: S / M / L / XL)-(shipping add $2)
[ ] _____ $170.35 School COMAL Package - 12 different books and 12 different disks (shipping add $7.20)

=====

Total_____ + _____ Shipping (minimum $2 per order) = Total Paid US$_____ (WI add 5% sales tax)

Mail To: COMAL Users Group USA, 6041 Monona Drive, Madison, WI 53716 or call 608-222-4432

Subscriber's Specials

This issue's subscriber-only specials are good only through January 25, 1985. If you are a current subscriber you MUST include your subscriber number with your order, or we will not honor the special prices. Your subscriber number is printed on the newsletter mailing label and all our invoices. New subscribers get these prices automatically and a number will be assigned with the order.

BACK ISSUES ONLY $1.50 EACH

Order anything at all, and get COMAL TODAY back issues for only $1.50 each. No extra shipping charge on UPS packages. First Class (Canada) add $1 extra per issue.

FREE DISK OFFER

*** Order User Group Disk #10 for $9.75 and get User Group Disk #11 free.
*** Order TODAY DISK #2 for $9.75 and get TODAY DISK #1 free.

DISK SUBSCRIPTION - ONLY $29.95

We could not find a lower priced disk subscription anywhere. Plus, our TODAY DISKS are full of programs on BOTH sides, almost like half price. Six disks (12 sides) cost only $29.95. Extend the subscription for more than 6 disks for only $5 per disk! You can even start the subscription back as far as TODAY DISK #1.

THE SHADOW KNOWS

99.9%

EFFECTIVE

Shadow is a new and revolutionary way to duplicate even your most protected software. It encompasses all the latest advances in software, as well as a highly sophisticated piece of hardware. This is absolutely **the best** utility available today. "It will even copy the other copy programs." Because of Shadow's unique abilities, we feel DOS protection is a thing of the past.

By the time you place your order we expect the Shadow to copy 100% — that's right, 100% — of **all** software available for the C-64.

Order by phone 24 hrs./7 days or send cashier's check/money order payable to Megasoft. Visa, MasterCard include card # and exp. date. Add $3.50 shipping/handling for continental U.S., $5.50 for UPS air. CODs add $7.50. Canada add $10.00 Other foreign orders add $15.00 and remit certified U.S. funds only. Distributors invited and supported.

MegaSoft LTD.

INTRODUCTORY OFFER
$89.95

P.O. Box 1080 Battle Ground, Washington 98604
Phone 1-800-541-1541 • BBS (206)687-5205 After-Hours Computer-to-Computer Ordering